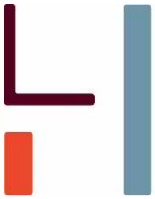
Monograph:
Marinho de Araujo, A.A. orcid.org/0000-0003-1419-4208 (2018) Statistical Analysis of Series of N-of-1 Trials Using R. Report. (Unpublished)

The University Of Sheffield.



LUXEMBOURG INSTITUTE OF HEALTH
RESEARCH DEDICATED TO LIFE

Statistical Analysis of Series of N-of-1 Trials Using R

Artur Araujo

September 2018

Acknowledgements

I would like to thank especially Dr. Stephen Senn for giving me the opportunity to work on the IDEAL project with him. I would like to acknowledge with thanks the helpful suggestions made by Dr. Stephen Senn and Dr. Steven A. Julious, during the revision of this document. Many thanks for the Luxembourg Institute of Health for receiving me so well during the first three years of my postgraduate studies and research. I would like to thank Boehringer Ingelheim GmbH for having paid my tuition fees at the University of Sheffield for the first three academic years. Special thanks to the Luxembourgish taxpayers and the Luxembourgish State through the CEDIES (Centre de Documentation et d'Information sur l'Enseignement Supérieur) for providing financial aid during my postgraduate studies at the Luxembourg Institute of Health and at the University of Sheffield. This project has received funding from the European Union's Seventh Framework Programme for research, technological development, and demonstration under grant agreement no 602552 (IDEAL - Integrated Design and Analysis of small population group trials).

Abstract

The statistical analysis of series of n-of-1 trials in which the treatments were randomized in cycles is described. Version 3.2 of the R statistical software is used for the analysis. It is not guaranteed that older versions of the R software produce equal results to the ones presented. Some of the code presented here might not run on older versions of the R software. Therefore, versions of R equal or superior to version 3.2 are recommended when running the code presented.

Table of Contents

1 Defining directories and reading data 1

2 Naïve estimation of individual treatment effects 5

3 The paired t-test 13

4 A summary measures approach 15

5 Analysis of variance 17

6 Preparing to use linear mixed-effects models 23

7 Linear mixed-effects model 25

7.1 Estimation and inference on the overall treatment effect 25

7.2 Estimation and inference on individual treatment effects 35

8 Linear mixed-effects model of difference 43

8.1 Estimation and inference on the overall treatment effect 44

8.2 Estimation and inference on individual treatment effects 46

9 Graphical methods 49

9.1 Scatterplots 49

9.2 Boxplots 66

9.3 Density plots 75

9.4 Dot plots 82

References 91

1 Defining directories and reading data

Before starting any analysis within R it is often useful to define the working directory to the one where all the files related to the analysis are placed. This way, relative paths instead of full paths can be used to tell R where the files are located. The "setwd" function with a character string defining the full path to the working directory as argument can be used.

```
> # define working directory
> while ( !"nof1_rand_cycles.csv" %in% list.files() ) {
+   file <- file.choose();# choose this file
+   WorkingDir  <- dirname(file);# get path to file
+   setwd(dir=WorkingDir); # define working directory
+   rm(file, WorkingDir); # remove objects
+ }
```

In this example, four statements are contained within a "while" loop. There is a condition in this "while" loop, where the existence of the file "nof1_rand_cycles.csv" is checked in the working directory. The purpose is to check that the working directory is correctly defined. If the file "nof1_rand_cycles.csv" is not present in the working directory, then the "file.choose" function is executed. The "file.choose" function prompts the user the select a file. After the user selects the file, the working directory is determined from the name of this file through the "dirname" function. Afterwards, the "setwd" function defines the working directory to the one determined in the previous statement. In the last statement from the "while" loop, the objects "file" and "WorkingDir" created in the previous statements, are removed from the R environment. After the four statements are executed, the condition from the "while" loop is checked again, and the loop is exited if the condition is false. If the condition is true, the process is repeated again. The loop may continue indefinitely, until the user selects a file contained within a directory, which itself contains the "nof1_rand_cycles.csv" file. The "nof1_rand_cycles.csv" file contains the data required for the analysis. When the "setwd" function succeeds, it does not return any visible value or error message. The "getwd" function can be called without any argument to query the current working directory and check that it is defined as desired:

```
> getwd();
[1] "E:/nof1_R_nalysis"
```

A single element character vector displaying the full path to the working directory is returned, confirming that the working directory is correctly defined.

The user can list all the files located within the working directory by calling the "list.files" function without arguments.

```
> list.files();
[1] "~$f1_rand_cycles.docx" "nof1_rand_cycles.csv"  "nof1_rand_cycles.docx"
[4] "nof1_rand_cycles.R"
```

The “list.files” function returns a character vector of the names of the files placed inside the working directory.

The data are stored within a “csv” file. To proceed with the analysis of the data, it must be imported to the R software. The “read.csv” function can be used to carry out this task:

```
> # import data
> ndata <- read.csv(
+   file="./nof1_rand_cycles.csv",
+   header=TRUE,
+   colClasses=c(
+     "factor", # Patient as factor
+     "factor", # Treatment as factor
+     "factor", # Cycle as factor
+     "factor", # Pair as factor
+     "numeric" # Y as numeric
+   )
+ );
```

In this example, the “file” parameter specifies the path to the file being imported. Both full and relative paths can be used. Here a relative path is used. By preceding the filename with the set of characters “./” one can define the path to the file as the working directory. When no path is specified, R looks into the working directory, so specifying the filename works as well. The “header=TRUE” parameter instructs the “read.csv” function to read the variable names from the header of the file into the R object. The colClasses argument defines the type of variable inside the R object being saved. In the present situation, five variables are defined being the first four of type “factor” and the last one of type “numeric”. Within the R software environment, categorical variables can be stored inside “factor” objects, and continuous or discrete ones can be stored inside “numeric” objects. The data is saved to a data.frame object named “ndata”.

The contents of a data.frame can be viewed by calling the “print” function with the data.frame name as argument. Since in this case the data.frame has a large number of rows, the output of the print would occupy a large space of this document, so it will not be displayed here. Instead, the “head” and “tail” functions can be used to display a desired number of rows of the “data.frame”. In the next example, the first six rows of the “ndata” data.frame are shown:

```
> head(ndata, n=6);
  Patient Treatment Cycle Pair    Y
1       1         A     1  1 1 2394
2       1         B     1  1 1 2686
3       1         A     2  1 2 2515
4       1         B     2  1 2 2675
5       1         A     3  1 3 2583
6       1         B     3  1 3 2802
```

The last six rows of the “ndata” object can be print by calling the “tail” function as in the example that follows:

```
> tail(ndata, n=6);
   Patient Treatment Cycle Pair    Y
67      12         A     1 12 1 2627
68      12         B     1 12 1 2759
69      12         A     2 12 2 2712
70      12         B     2 12 2 2698
71      12         A     3 12 3 2572
72      12         B     3 12 3 2826
```

Listing the structure of the data.frame can be very informative for the data analyst. This can be achieved by calling the “str” function with the desired data.frame name as argument.

```
> str(ndata);
'data.frame':    72 obs. of  5 variables:
 $ Patient  : Factor w/ 12 levels "1","10","11",..: 1 1 1 1 1 1 5 5 5 5 ...
 $ Treatment: Factor w/ 2 levels "A","B": 1 2 1 2 1 2 1 2 1 2 ...
 $ Cycle    : Factor w/ 3 levels "1","2","3": 1 1 2 2 3 3 1 1 2 2 ...
 $ Pair     : Factor w/ 36 levels "1 1","1 2","1 3",..: 1 1 2 2 3 3 13 13 14 14 ...
 $ Y        : num  2394 2686 2515 2675 2583 ...
```

This output gives the following useful information. The “ndata” object is a data.frame object, which contains five variables with 72 observations each. The variable “Patient” is a factor with 12 levels; the variable “Treatment” is a factor with two levels; the variable “Cycle” is defined as a factor with three levels; the variable “Pair” is a factor with 36 levels; and finally the variable “Y” is a numeric one. Knowing that the data was collected from a series of n-of1 trials, from this information the data analyst can deduce that two treatments were randomized in a maximum of 3 cycles within a total of 12 patients. It is obvious that the outcome variable is identified by “Y”. It can be observed that the “Pair” variable results from the concatenation of the “Patient” and “Cycle” variables, so it can be considered redundant in this data set.

2 Naïve estimation of individual treatment effects

One of the purposes of running clinical trials is to estimate a mean treatment difference. For a series of n-of-1 trials design where two treatments are randomized in cycles, if there is no missing information, there should be an outcome observation for each of the two treatments administered to each individual under each cycle. In such a case, a new outcome variable can be obtained by differencing the outcome variable registered under both treatments for each cycle. A new dataset can be obtained with half as many observations as the original dataset. A proper statistical analysis can be carried out on the recoded smaller dataset. Some statistical methods give exactly the same results when applied to each of the mentioned datasets, if the necessary modifications are done to these methods.

Before proceeding with the dataset recoding, the levels of the "Patient" factor are reordered in the original dataset. Levels of a factor can be any character string. Since in this case the factor levels are actual numbers encoded as character strings, the factor levels can be ordered. In this case, the factor levels are ordered by increasing order. The next lines of code accomplish this.

```
> # reorder factor levels on original dataset
> ndata$Patient <- factor(
+   x=ndata$Patient,
+   levels=levels(ndata$Patient)[
+     order( as.numeric( levels(ndata$Patient) ) )
+   ]
+ );
```

Note that changing the order of the factor levels within a data.frame modifies an attribute of the factor but does not modify the data contained within. Reordering the levels of the "Patient" factor ensures that the levels are processed in the desired order in subsequent code. The usefulness of this reordering of factor levels is more evident below where plots of the data are demonstrated.

Next, the R code that leads to the smaller dataset is presented without entering into details as regards each line of code.

```
> # compute differences
> ncycles <- nrow( unique(ndata[,c("Patient", "Cycle")]) ); # cycle number
> ddata <- data.frame(
+   "Patient"=numeric(ncycles),
+   "Cycle"=numeric(ncycles),
+   "YA"=numeric(ncycles),
+   "YB"=numeric(ncycles),
+   "dY"=numeric(ncycles)
+ );
>
> index <- 1;
> for ( patient in levels(ndata$Patient) ) { # patients loop
+   for ( cycle in levels(ndata$Cycle) ) { # cycles loop
+     indexA <- with(
+       ndata,
+       which(
+         Patient==patient
+         & Cycle==cycle
+         & Treatment==levels(Treatment)[1]
+       )
+     );
+     indexB <- with(
+       ndata,
+       which(
+         Patient==patient
+         & Cycle==cycle
+         & Treatment==levels(Treatment)[2]
+       )
+     );
+     if (length(indexB)==0 & length(indexA)==0) next; # unbalanced data
+     if (length(indexB)==0) {
+       warning(
+         "An observation under subject ",
+         patient,
+         ", cycle ",
+         cycle,
+         " and treatment ",
+         levels(ndata$Treatment)[2],
+         " is not available to define a point!\nIgnoring."
+       );
+       next; # next loop
+     }
+     else if (length(indexA)==0) {
+       warning(
+         "An observation under subject ",
+         patient,
+         ", cycle ",
+         cycle,
+         " and treatment ",
+         levels(ndata$Treatment)[1],
+         " is not available to define a point!\nIgnoring."
+       );
```

```
+       next; # next loop
+     }
+     ddata$Patient[index] <- patient;
+     ddata$Cycle[index] <- cycle;
+     ddata$YA[index] <- ndata$Y[indexA];
+     ddata$YB[index] <- ndata$Y[indexB];
+     ddata$dY[index] <- ndata$Y[indexB]-ndata$Y[indexA];
+     index <- index+1; # increase index for next loop
+   }
+ }
>
> # coerce 'Patient' and 'Cycle' within 'ddata' to factor
> ddata <- within(
+   ddata,
+   {
+     Patient <- factor(Patient);
+     Cycle <- factor(Cycle);
+   }
+ );
>
> # reorder factor levels on recoded dataset
> ddata$Patient <- factor(
+   x=ddata$Patient,
+   levels=levels(ddata$Patient)[
+     order( as.numeric( levels(ddata$Patient) ) )
+   ]
+ );
```

At this stage, it is important to remind the reader that comments in R code are preceded by a hash "#" character. Comments are not interpreted by the R software while running the code. After running the above code, the recoded data.frame named "ddata" is obtained and accessible for subsequent use. The first six rows of "ddata" contain data on the first two patients.

```
> head(ddata, n=6);
  Patient Cycle   YA   YB  dY
1       1     1 2394 2686 292
2       1     2 2515 2675 160
3       1     3 2583 2802 219
4       2     1 2746 2726 -20
5       2     2 2592 2867 275
6       2     3 2743 2742  -1
```

In the recoded data.frame, five variables can be identified. There is a categorical variable indicating the patient and a categorical variable indicating the cycle and then three continuous variables, one each for outcome under treatments A and B and one for the difference. The column "YA" contains the values of outcome variable measured under treatment labelled as "A", and the column "YB" contains the values of outcome variable registered under treatment "B" in the original dataset. The column

"dY" is obtained as the difference of the elements of the column "YB" and the elements of the column "YA" in the same row. The variables "YA" and "YB" are retained for informative purposes, and should not be needed for the statistical analysis of the recoded dataset. Note that cycles should be regarded as being 'nested within' patients. That is to say, that although the same cycle numbers appear for every patient there is no implication that cycle 3 for patient 1 is the same as cycle 3 for patient 2. To refer to a given cycle it is necessary to refer not only to the cycle number but also to the patient.

The number of observations can be read from the output given by the "str" function as shown above. Since there are as many observations as rows in "ddata", the number of observations can alternatively be obtained from the "nrow" function.

```
> nrow(ddata);
[1] 36
```

As mentioned above the recoded dataset contains half the observations of the full dataset. There are 72 observations in the original dataset and 36 in the recoded one.

The means of the outcome difference under the two treatments can be obtained for each patient as follows:

```
> # summary contrasts per patient
> sddata <- by(
+   data=ddata$dY,
+   INDICES=ddata$Patient,
+   FUN=mean
+ );
>
> # display per patient mean of outcome difference
> print(sddata);
ddata$Patient: 1
[1] 223.6667
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 2
[1] 84.66667
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 3
[1] 60
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 4
[1] 348
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 5
[1] 259.3333
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 6
[1] 50
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 7
[1] 175
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 8
[1] 153.6667
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 9
[1] 324.3333
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 10
[1] 247.6667
------------------------------------------------------------------------------------------
----------------------------------
ddata$Patient: 11
[1] 214.3333
```

```
------------------------------------------------------------------------------
--------------------------------
ddata$Patient: 12
[1] 124
```

At first the “by” function applies the “mean” function to the variable “dY” over each factor of the “Patient” variable contained within the “ddata” data.frame. The “by” function returns an R object that is saved under the name “sddata”. Afterwards the “print” function prints the contents of “sddata” on screen. It can easily be observed that the minimum per patient mean difference values range between 50 and 348 units. For datasets with large numbers of individuals, this can become hard to identify. Fortunately, there is the “range” function:

```
> range(sddata);
[1]  50 348
```

The “which.min” and “which.max” functions can be used to determine which index elements of a vector contain the minimum and maximum values respectively. Since in this case the levels of the “Patient” factor are ordered and equal to the index of “ddata”, these functions can help determine which patients registered the minimum mean and the maximum mean.

```
> # which patient registered the minimum mean
> which.min(sddata);
6
6
>
> # which patient registered the maximum mean
> which.max(sddata);
4
4
```

The answer is that the minimum mean difference of outcome was registered for patient number 6, and the maximum mean difference of outcome was registered for patient identified by number 4. Here two lines with equal numbers are returned. The first line is the name of the vector element and the second line is the index of the vector element where either the minimum or the maximum values are observed inside the vector given as argument. Given that, the elements of vector “sddata” are named after the patient labels that are identified by numbers in increasing order in the dataset from which this vector was computed, the vector element names are equal to the vector indexes in this case.

The minimum, first quantile, median, mean, third quantile and maximum can be obtained from a single function call.

```
> summary(sddata)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   50.0   114.2   194.7   188.7   250.6   348.0
```

The individual means of outcome variable difference are shortly referred in the n-of-1 trial literature as individual treatment effects. The method of estimation of the individual treatments effects that is described in this section can be considered a simple and naïve one, and is not recommended. The individual treatment effects can be estimated more precisely using more advanced statistical techniques that are presented below.

3 The paired t-test

The paired t-test assumes the data as independent and normally distributed. The independency assumption is questionable for any measurement taken from the same individual as is done for n-of-1 trials. However, a justification can be provided in terms of testing the point null hypothesis that the treatments are identical for any individual. In that case, under the null hypothesis at least and given randomisation (which will vary the order of the A and B treatments) the differences per pair can be treated as if they were independent. The t-test can be performed on both datasets presented above. The following example considers a two sided paired t-test performed on the full dataset.

```
> t.test(
+    formula=Y~Treatment,
+    data=ndata,
+    alternative="two.sided",
+    mu=0,
+    var.equal=TRUE,
+    conf.level=0.95
+ );

        Paired t-test

data:  Y by Treatment
t = -7.111, df = 35, p-value = 2.749e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -242.6002 -134.8443
sample estimates:
mean of the differences
              -188.7222
```

In this function call, the “data” argument inputs the “ndata” data.frame to the “t.test” function. The “formula” argument tells the function to perform the test on the values of the variable “Y” for each factor of the variable “Treatment”. The purpose here is to compare the means of the outcome variable under the two treatments labelled “A” and “B” in the data.frame. By specifying the argument “mu” equal to zero and a two sided test as specified by argument “alternative”, the alternative hypothesis is defined as “true difference in means is not equal to 0” as printed in the output. In this case, the null hypothesis is defined, as “true difference in means is equal to zero”. The “paired=TRUE” specifies a paired t-test. The “conf.level” argument defines both the level of significance for the test and the coverage of the confidence interval of the mean of differences. Here a 95% confidence interval and a 5% level of significance are specified. The p-value is significantly lower than the 0.05 level of

significance, leading to the rejection of the null hypothesis. There is statistical evidence that the treatments are significantly different when considering all the patients recruited into the trial.

The paired t-test can be performed on the smaller dataset where the outcome is defined as the difference of outcome under each of the two treatments studied. Now the cycles define the pair.

```
> t.test(x=ddata$dY, alternative="two.sided", mu=0, conf.level=0.95);

        One Sample t-test

data:  ddata$dY
t = 7.111, df = 35, p-value = 2.749e-08
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 134.8443 242.6002
sample estimates:
mean of x
 188.7222
```

Unlike in the previous example there is no need to specify the “paired” parameter. The results are similar to the results of the t-test performed on the full dataset. The difference resides in the signs of the t statistic, the estimate and the lower and upper bounds of the confidence interval. The results are however the same in absolute value. In the t-test performed on the full dataset the mean of outcome under treatment “A” minus the mean of outcome under treatment “B” is estimated. While for the t-test when performed on the recoded dataset, the mean of outcome under treatment “B” comes before the mean of outcome under treatment “A” in the difference. The p-value obtained is the same. Moreover, the conclusions are the same as above.

4 A summary measures approach

An alternative consists in performing the t-test on the summary data saved earlier under the R object named “sddata”. In this example, the same alternative hypothesis as the one exemplified in section 3 is tested. To compare the results the significance level is kept at 0.05.

```
> t.test(x=sddata, alternative="two.sided", mu=0, conf.level=0.95);

        One Sample t-test

data:  sddata
t = 6.649, df = 11, p-value = 3.616e-05
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 126.2500 251.1945
sample estimates:
mean of x
 188.7222
```

The degrees of freedom are equal to 11 instead of 35. This is due to the lower number of observations present in the latter dataset. There is one observation per patient in this dataset. The mean estimate is the same as in the second example presented in section 3. However, the bounds of the confidence interval of the mean are wider suggesting a larger standard error of the mean. The p-value is higher, but is still very low when compared to the significance level. This result suggests the rejection of the null hypothesis and the same conclusion as in the t-test examples presented above.

An advantage of this approach is as regards the calculation of the confidence interval. As mentioned before, under the strict null hypothesis that the treatments are identical for every patient, the matched pairs t-test is valid. However, as soon as one considers the possibility that the treatment effect is not zero, which is what is done by calculating a confidence interval, then the strict null is abandoned. It becomes plausible to believe that the treatment effect might vary from patient to patient. The summary measures approach makes allowance for this.

5 Analysis of variance

In the next example, an analysis of variance considering the cycle within patients as block structure is performed on the full dataset.

```
> aov0 <- aov(formula=Y~Treatment+Error(Patient/Cycle), data=ndata);
> print(aov0);

Call:
aov(formula = Y ~ Treatment + Error(Patient/Cycle), data = ndata)

Grand Mean: 2720.111

Stratum 1: Patient

Terms:
                Residuals
Sum of Squares    1458791
Deg. of Freedom        11

Residual standard error: 364.1667

Stratum 2: Patient:Cycle

Terms:
                Residuals
Sum of Squares   316884.7
Deg. of Freedom        24

Residual standard error: 114.9066

Stratum 3: Within

Terms:
                Treatment Residuals
Sum of Squares   641089.4  443735.6
Deg. of Freedom         1        35

Residual standard error: 112.5973
Estimated effects are balanced
```

Note how the Cycle within Patient is specified in the “Error” term of the formula parameter. Here the purpose is to test the difference of the mean of outcome under the two treatments, hence the Treatment term outside the “Error” term of the formula parameter. The “aov” function returns an object that is saved under the name “aov0”. Then this object can be used as argument to other functions as is seen next. The “summary” of “aov0” follows.

```
> summary(aov0);

Error: Patient
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals 11 1458791  132617

Error: Patient:Cycle
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 24 316885   13204

Error: Within
          Df Sum Sq Mean Sq F value    Pr(>F)
Treatment  1 641089  641089   50.57 2.75e-08 ***
Residuals 35 443736   12678
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

As can be seen from the output, a table containing the sum of squares, the mean of squares and the degrees of freedom for each error term is displayed on screen. Note the same degrees of freedom and p-value as in the paired t-test and equivalent one sample t-test exemplified above. Note also that the F statistic obtained here equals the square of the t statistic obtained in the referred t-tests. The F test considers a null hypothesis of equality of the means estimated under the two factors against an alternative hypothesis of difference of the means. The p-value resulting from the F test is significantly lower than the usual 0.05 significance level, suggesting that the means estimated under the two treatments are different. One treatment should be preferred over the other. The table of means indicates which treatment should be preferred.

```
> print(model.tables(x=aov0, type="means"), digits=3);
Tables of means
Grand mean

2720.111

 Treatment
Treatment
   A    B
2626 2814
```

Therefore, if the patient feels better when the outcome variable registers higher values, the treatment “B” should be preferred over treatment “A”. If it is the case that the quality of life of the patient is better when lower values of outcome variable are observed, then the choice should fall over treatment “A”.

The following example prints the coefficients under each treatment and the standard deviation.

```
> print(model.tables(x=aov0, type="effects", se=TRUE), digits=3);
Tables of effects

 Treatment
Treatment
    A     B
-94.4  94.4

Standard errors of effects
        Treatment
            18.8
replic.       36
```

The analysis of variance considering the cycle and treatment within patient can be performed by adding the Treatment variable to the “Error” term as in the following example.

```
> aov1 <- aov(
+   formula=Y~Treatment+Error( Patient/(Treatment*Cycle) ),
+   data=ndata
+ );
> print(aov1); # print results

Call:
aov(formula = Y ~ Treatment + Error(Patient/(Treatment * Cycle)),
    data = ndata)

Grand Mean: 2720.111

Stratum 1: Patient

Terms:
                Residuals
Sum of Squares    1458791
Deg. of Freedom        11

Residual standard error: 364.1667

Stratum 2: Patient:Treatment

Terms:
                Treatment Residuals
Sum of Squares   641089.4  159516.3
Deg. of Freedom         1        11

Residual standard error: 120.4221
Estimated effects are balanced

Stratum 3: Patient:Cycle

Terms:
                Residuals
Sum of Squares   316884.7
Deg. of Freedom        24

Residual standard error: 114.9066

Stratum 4: Patient:Treatment:Cycle

Terms:
                Residuals
Sum of Squares   284219.3
Deg. of Freedom        24

Residual standard error: 108.8231
```

Again, the p-value for testing the difference between the means of outcome estimated under the two treatments can be accessed through the “summary” function.

```
> summary(aov1);

Error: Patient
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals 11 1458791  132617

Error: Patient:Treatment
          Df Sum Sq Mean Sq F value    Pr(>F)
Treatment  1 641089  641089   44.21 3.62e-05 ***
Residuals 11 159516   14501
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Error: Patient:Cycle
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 24 316885   13204

Error: Patient:Treatment:Cycle
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals 24 284219   11842
```

It can be noted that the p-value and degrees of freedom obtained in this example are the same as the ones obtained from the one sample t-test of the summary measures data presented above. While the obtained F statistic is equal to the square of the t statistic obtained in the summary measures approach. The p-value remains relatively low and the conclusion drawn from the above analysis of variance table remains.

The overall mean and the mean under each treatment rounded to three decimal places follow.

```
> print(model.tables(x=aov1, type="means"), digits=3);
Tables of means
Grand mean

2720.111

 Treatment
Treatment
   A    B
2626 2814
```

Note that the mean values are the same as the ones resulting from the analysis of variance without the treatment by patient interaction in the “Error” term.

The standard error of the Treatment dummy variable coefficient is obtained next.

```
> print(model.tables(x=aov1, type="effects", se=TRUE), digits=3);
Tables of effects

 Treatment
Treatment
    A     B
-94.4  94.4

Standard errors of effects
        Treatment
             20.1
replic.        36
```

Note a slight increase in the standard error.

6 Preparing to use linear mixed-effects models

Linear mixed-effects models can be fitted within R after loading the required packages [1-3]. Both the “lme” function provided by “nlme” package and the “lmer” function provided by “lme4” package can be used [2, 3]. The “nlme” package is included with the R base distribution available for the Microsoft Windows operating system. Function “lmer” provides a flexible way of specifying the formula parameter that is unavailable in the “lme” function, so “lmer” function from package “lme4” is used to fit linear mixed-effect models in all the examples given below. Package “lme4” can be downloaded and installed by calling “install.packages” from the R console. It should work when a viable internet connection is available. The output may vary depending on the operating system and particular system configuration. The following output was obtained on the test machine.

```
> # install package if not installed
> if ( !"lme4" %in% installed.packages() ) {
+   install.packages("lme4");
+ }
Installing package into ‘C:/Users/aaraujo/Documents/R/win-library/3.2’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.2/lme4_1.1-10.zip'
Content type 'application/zip' length 4787883 bytes (4.6 MB)
downloaded 4.6 MB

package ‘lme4’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aaraujo\AppData\Local\Temp\RtmpUF4Arz\downloaded_packages
```

The code checks if package “lme4” is installed, and if not then “install.packages” is called in order to install the package.

The covariance matrix of the fixed effects and the likelihood ratio tests that are currently implemented in package “lme4” are based on asymptotic approximations. Package “pbkrtest” provides Kenward-Roger and parametric bootstrap based methods for linear mixed-effects model comparison [4, 5]. These two methods do not rely on asymptotic approximations, therefore being expected to behave better for smaller samples. Package “pbkrtest” was developed as an extension to package “lme4”, so the functions implemented in package “pbkrtest” expect objects returned by “lmer” function from package “lme4”. If package “pbkrtest” is not already available on the local machine, it can be downloaded and installed as follows.

```
> # install package if not installed
> if ( !"pbkrtest" %in% installed.packages() ) {
+   install.packages("pbkrtest");
+ }
Installing package into ‘C:/Users/aaraujo/Documents/R/win-library/3.2’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.2/pbkrtest_0.4-2.zip'
Content type 'application/zip' length 205618 bytes (200 KB)
downloaded 200 KB

package ‘pbkrtest’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\aaraujo\AppData\Local\Temp\RtmpELbccB\downloaded_packages
```

To access all the functions implemented for linear mixed-effects model analysis, the “lme4” and “pbkrtest” packages must be loaded into the R software. Some of the functions presented below in this document use parallel computation, so package “parallel” must be loaded as well [6]. The “library” function with the package name as argument can be used to accomplish this task.

```
> # load lme4 package
> library(lme4);
Loading required package: Matrix
>
> # load pbkrtest package
> library(pbkrtest);
>
> #load parallel package
> library(parallel);
```

7 Linear mixed-effects model

After installing and loading the required packages, the mixed-model analysis can commence. A model with the Treatment as fixed and a Patient random term, a random Treatment by Patient interaction and a random Cycle by Patient interaction, is fitted in the next example [1, 7-9].

```
> fit0 <- lmer(
+   formula=Y~Treatment+(1|Patient)+(1|Patient:Cycle)+(1|Patient:Treatment),
+   data=ndata,
+   REML=TRUE
+ );
> print(fit0);
Linear mixed model fit by REML ['lmerMod']
Formula: Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
   Data: ndata
REML criterion at convergence: 893.7924
Random effects:
 Groups             Name        Std.Dev.
 Patient:Cycle      (Intercept)  26.09
 Patient:Treatment  (Intercept)  29.77
 Patient            (Intercept) 139.50
 Residual                       108.82
Number of obs: 72, groups:  Patient:Cycle, 36; Patient:Treatment, 24; Patient, 12
Fixed Effects:
(Intercept)    TreatmentB
     2625.8         188.7
```

The model is fitted on the full dataset. The “REML=TRUE” argument instructs the function to fit the model using restricted maximum likelihood, a method of estimation that provides unbiased estimates of the variance parameters of the model. In this example, the “lmer” function returns an object that is saved under the name “fit0”. Afterwards the “fit0” object is printed on screen. Here the coefficients of the fixed effects part of the model and the standard deviations pertaining to the random part of the model are displayed.

7.1 Estimation and inference on the overall treatment effect

The fixed effects coefficients can also be extracted from the “fit0” object using the “fixef” function.

```
> fixef(fit0);
(Intercept)    TreatmentB
  2625.7500     188.7222
```

Therefore, the estimated intercept of the model equals 2625.7500 units of outcome variable. The estimate of the difference of the means of outcome under treatment “B” and treatment “A” assumes the value 188.7222 units. This quantity is also shortly referred to in the medical statistics literature as the overall treatment effect.

The variance components of the linear mixed-effects model can be saved under an object returned from a call to “VarCorr” with object “fit0” as argument.

```
> vfit0 <- VarCorr(fit0);
```

Then the “vfit0” object as returned from “VarCorr” can be displayed on screen for analysis. To display the variances and standard errors, a call to “print” can be run as in the following example.

```
> print( x=vfit0, comp=c("Variance", "Std.Dev") );
 Groups              Name        Variance Std.Dev.
 Patient:Cycle       (Intercept)   680.53  26.087
 Patient:Treatment   (Intercept)   886.34  29.771
 Patient             (Intercept) 19459.14 139.496
 Residual                        11842.47 108.823
```

The variance and standard error of the cycle by patient, the treatment by patient, the patient and the residual errors are displayed in this order.

The covariance matrix of the fixed effects plays an important role when making inferences on these. It can be computed and displayed through a call to “vcov” with an object returned by “lmer” function as argument.

```
> vcov(fit0);
2 x 2 Matrix of class "dpoMatrix"
             (Intercept) TreatmentB
(Intercept)    2043.3177  -402.8189
TreatmentB     -402.8189   805.6377
```

Of particular interest is the variance of the difference in means obtained under each treatment, which in the present case equals 805.6377 units. At the time of writing this document, “vcov” returns an asymptotic approximation of the covariance of the fixed effects, which can be biased for unbalanced data, but is unbiased for balanced data. The dataset used as example in this document is balanced for the variables “Patient and Treatment within Patient. This means that all the levels of the factor “Patient” have an equal number of observations. Moreover, all the levels of the “Treatment” factor have the same number of observations for each level of factor “Patient”. An alternative approach was developed by Kenward and Roger that attempts to minimize the bias of the covariance matrix of the fixed effects in a mixed-effects model [5]. It can be obtained by calling the “vcovAdj” function with an object returned by function “lmer” as argument.

```
> vcovAdj(fit0);
2 x 2 Matrix of class "dgeMatrix"
             (Intercept) TreatmentB
(Intercept)   2043.3177  -402.8189
TreatmentB    -402.8189   805.6377
```

The dataset used to fit the mixed-effects model is balanced so the Kenward-Roger covariance matrix of the fixed effects is exactly equal to the asymptotic one. For unbalanced data, the Kenward-Roger and asymptotic results are expected to differ.

To make inferences on the mean of outcome difference between the two treatments one must proceed by fitting two models, one with the Treatment factor in the fixed effects term and another without. The two models must be fitted by maximum likelihood instead of restricted maximum likelihood. Both models can then be compared by the likelihood ratio test. Likelihood ratio tests of nested models with different fixed-effects fitted by restricted maximum likelihood are not meaningful. For this reason, maximum likelihood is used to fit both models.

```
> # Model with Treatment
> fit00 <- refitML(fit0);
>
> # Model without Treatment
> fit01 <- update(fit00, formula=~.-Treatment);
>
> # Likelihood ratio test
> anova(fit00, fit01);
Data: ndata
Models:
fit01: Y ~ (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
fit00: Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
      Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
fit01  5 940.95 952.33 -465.47   930.95
fit00  6 923.59 937.25 -455.80   911.59 19.359      1  1.083e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

One starts by fitting a model with the intercept and the Treatment in the fixed effect part, a Patient, Cycle by Patient interaction and a Treatment by Patient interaction in the random effects part. Such a model has been previously fitted by restricted maximum likelihood and an object exists under the name “fit0”. It suffices to refit the model by maximum likelihood with function “refitML”. The object resulting from the maximum likelihood fit is saved under the name “fit00”. A second model is fitted with the same fixed and random effect terms as the previous one, except the Treatment in the fixed effects term. The easiest way to accomplish this in R is to use the “update” function to update the “fit00” object with the variable being removed in the formula. Note the dot “.” character in the formula argument. The dot “.” character represents all the terms present in the formula of the object

being updated, in this case “fit00”. By placing the minus “-” character and the desired variable after the dot “.” character in the formula argument, a new model without that variable is fitted to the same dataset. The “update” function returns an object saved under the name “fit01”. Finally, a call to “anova” is made with objects “fit00” and “fit01” as arguments. The “anova” function returns an object that is printed on screen displaying a table with the results of the likelihood ratio test. The likelihood ratio statistic equals twice the difference between the log likelihoods of the full model and the reduced model. Under the null hypothesis, it follows asymptotically a chi square distribution with a number of degrees of freedom equal to the difference between the degrees of freedom of the full model and the reduced one. In the case at hand, the likelihood ratio statistic equals 19.359 units with one single degree of freedom. The probability of getting a likelihood ratio statistic that is superior to the observed one is the p-value. A very low p-value indicates that the log likelihood of the full model is significantly superior to the log likelihood of the reduced model. In other words, the full model fits better to the data than the reduced one. At a significance level of 0.05, the null hypothesis that the reduced model fits better to the data than the full model should be rejected. The full model is preferred over the reduced one. Here the likelihood ratio test suggests that the coefficient of the Treatment factor should be retained and that the treatments being compared are significantly different one from the other. So one treatment should be preferred over the other. The choice of treatment now depends on the order of the levels of the Treatment factor in the data, the sign of the coefficient of the Treatment factor in the model and on the preference of the patient for higher or lower values of outcome variable.

The likelihood ratio test provided by the “anova” function is based on the asymptotic chi-square distribution of the test statistic, and as such can be considered an approximation when applied to small samples. A parametric bootstrap approach simulates the distribution of the test statistic. As such is expected to result in a more accurate p-value depending on the number of bootstrap simulations. Being that higher numbers of bootstrap simulations provide results that are more accurate. The following example performs a parametric bootstrap likelihood ratio test between the full model and the model reduced by removing the “Treatment” factor.

```
> # seed for RNG
> iseed <- rep(x=12345, times=6);
>
> # create cluster
> cl <- makeCluster(spec=rep("localhost", 2), type="PSOCK");
>
> # set RNG seed on cluster
> clusterSetRNGStream(cl=cl, iseed=iseed);
>
> # compare models using parametric bootstrap method
> PBmodcomp(largeModel=fit00, smallModel=fit01, nsim=10000, cl=cl);
Parametric bootstrap test; time: 186.61 sec; samples: 10000 extremes: 0;
large : Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
small : Y ~ (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
         stat df    p.value
LRT    19.359  1 1.083e-05 ***
PBtest 19.359    9.999e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
> # stop cluster
> stopCluster(cl=cl);
```

Five lines of code are run in this example. At first, a vector with six elements is created and saved under the name “iseed”. This vector provides the seed for the random number generator used to simulate the bootstrap samples. Afterwards a parallel cluster with two R sessions named “cl” is created. Here the “makeCluster” function opens two R sessions on the local machine to be run in parallel. In this case, if the central processing unit present on the local machine has at least two cores, the two R sessions will run one on each core in parallel. In the third line, the seed for the random number generator is set on the cluster of parallel R sessions. In the fourth line, the “PBmodcomp” function is called to perform the parametric bootstrap comparison between the mentioned full and reduced models. In the “PBmodcomp” function, the “largeModel” parameter specifies the R object holding the full model; the “smallModel” argument defines the R object holding the reduced model; the “nsim” parameter is used to specify the number of bootstrap simulations; and the “cl” argument provides the cluster for parallel computation. Finally, the “stopCluster” function ends all the running R sessions present in the cluster supplied as argument. A seed is used to make sure that anyone running these very same lines of code obtains exactly the same results displayed here. The number of bootstrap simulations performed is unusually high so the results can be considered accurate enough. In the output of the “PBmodcomp” function, the results of the approximate likelihood ratio test and the parametric bootstrap likelihood ratio test are displayed. The p-values obtained from both methods differ but the conclusions remain the same as before.

Comparison of nested models can also be performed through an F-test based on a Kenward-Roger approximation.

```
> KRmodcomp(largeModel=fit00, smallModel=fit01);
F-test with Kenward-Roger approximation; computing time: 0.11 sec.
large : Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
small : Y ~ (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
        stat     ndf     ddf F.scaling   p.value
Ftest 44.209   1.000  11.000         1 3.616e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The F-test method presented here requires that both models being compared share the same covariance structure, so it cannot be used to compare nested models with distinct random effect terms. Unlike in the likelihood ratio test both models can be fit through restricted maximum likelihood. Note that the test statistic and the p-value are very close to the values obtained from the analysis of variance considering the cycle and treatment within patient presented above. The p-value obtained leads to the conclusion that the two treatments are significantly different one from the other, as explained numerous times above.

The significance of the random effect terms of the model can also be tested by means of likelihood ratio tests. To test the significance of the random effects of the original model fit by maximum likelihood, the “update” function can be used to refit new models without the desired random effects terms.

```
> # remove Treatment by Patient interaction
> fit02 <- update( fit00, formula=~.-(1|Patient:Treatment) );
>
> # remove Cycle by Patient interaction
> fit03 <- update( fit02, formula=~.-(1|Patient:Cycle) );
>
> # Likelihood ratio test
> anova(fit00, fit02, fit03);
Data: ndata
Models:
fit03: Y ~ Treatment + (1 | Patient)
fit02: Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle)
fit00: Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
      Df     AIC     BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
fit03  4 919.68 928.79 -455.84   911.68
fit02  5 921.64 933.03 -455.82   911.64 0.0342      1     0.8533
fit00  6 923.59 937.25 -455.80   911.59 0.0541      1     0.8161
```

In the first line of code presented on the last example, a call to “update” is made with the object relative to the full model as argument. The formula argument instructs the function to refit a new

model without the treatment by patient interaction random term. The object returned is saved under the name “fit02” for later use. On the second line of code a new mixed effects model refit is done on “fit02” where the random cycle by patient interaction term is removed. Therefore, “fit03” is an object relative to a further reduced model without both the treatment by patient interaction and the cycle by patient interaction random terms. Finally, the three models are compared by the likelihood ratio test through a call to “anova” function, where the objects relative to the three nested models are used as arguments. A table with four rows including the header row is displayed on screen. The fourth row in the table contains the results of the comparison between the full model, saved under “fit00”, and the model reduced by removing the treatment by patient interaction random term saved under “fit02”. The third row contains the results of the comparison of the later model contained in object “fit02”, and the model without both interaction random terms contained in object “fit03”. By successively removing, each of the interaction random error terms there is a very small decrease in the log likelihood, which implies very small likelihood ratio statistics and very high p-values. One can conclude that the treatment by patient interaction and the cycle by patient interaction are both non-significant. Despite the observed non-significance of the cycle by patient interaction, it reflects the randomization procedure used in the trial, so there is interest in keeping it in the model. The treatment by patient interaction implies important design considerations and is of particular interest for series of n-of-1 trials. It should not in any case be left out of a linear mixed-effects regression model when the purpose is to apply that model to data arising from a series of n-of-1 trials.

Confidence intervals for the parameters of the model can be obtained through a call to “confint” with an object returned by an “lmer” fit as first parameter.

```
> confint(fit0, level=0.95);
Computing profile confidence intervals ...
                  2.5 %     97.5 %
.sig01          0.00000   83.21930
.sig02          0.00000   85.94658
.sig03         83.69249  220.42530
.sigma         84.02994  135.87619
(Intercept) 2534.84191 2716.65809
TreatmentB   130.89839  246.54606
Warning message:
In optwrap(optimizer, par = start, fn = function(x) dd(mkpar(npar1,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
```

Here 95 percentage confidence intervals for the parameters of the linear mixed effects model are displayed. The “level” argument accepts any value between zero and unity to specify the desired confidence level. The “confint” function has an argument “method” to define one of three possible methods of estimation. When “method” is not specified, profile confidence intervals are estimated

and displayed. In this case, a table with five rows and two columns is displayed. There is a row for each parameter of the model and the first column contains the lower bound of the confidence interval, being the upper bound located in the second column. In the last row of the table, the confidence interval for the mean of difference between the treatments is located. The fact that the confidence interval for the mean of difference does not contain zero is consistent with the significance of the respective coefficient found earlier. The row identified by "(Intercept)" contains the confidence interval for the intercept of the model. The ".sigma" row is relative to the confidence interval for the standard error of the residual random term. The identification of the remaining three rows becomes somewhat difficult. One must criticize package "lme4" developers for their choice of row names here. Without any information provided, users must act as detectives and try to guess which row relates to which parameter. There is the possibility that the parameters are displayed here in the same order as they are displayed in the print of the model object.

```
> print(fit0);
Linear mixed model fit by REML ['lmerMod']
Formula: Y ~ Treatment + (1 | Patient) + (1 | Patient:Cycle) + (1 | Patient:Treatment)
   Data: ndata
REML criterion at convergence: 893.7924
Random effects:
 Groups            Name        Std.Dev.
 Patient:Cycle     (Intercept)  26.09
 Patient:Treatment (Intercept)  29.77
 Patient           (Intercept) 139.50
 Residual                      108.82
Number of obs: 72, groups:  Patient:Cycle, 36; Patient:Treatment, 24; Patient, 12
Fixed Effects:
(Intercept)    TreatmentB
     2625.8         188.7
```

So following the mentioned possibility, row ".sig01" would contain the confidence interval for the standard error of the cycle by patient interaction. Row ".sig02" would contain the confidence interval for the standard error of the treatment by patient interaction. The last row that remains to be identified ".sig03" would contain the confidence interval for the standard error of the patient random term in the model. Confidence intervals displayed in rows ".sig01" and ".sig02" contain the value 0, suggesting the non-significance of their respective random terms. A fact that is consistent with the conclusions of the analysis of variance and the likelihood ratio tests presented above. Therefore, it is very likely that the rows were correctly attributed to the model parameters. The relative magnitudes of the estimates of the standard errors of the random terms of the model also point in that direction.

Approximate confidence intervals for the fixed effects parameters of the model can be obtained by the "Wald" method.

```
> confint(fit0, level=0.95, method="Wald");
                 2.5 %    97.5 %
.sig01              NA        NA
.sig02              NA        NA
.sig03              NA        NA
.sigma              NA        NA
(Intercept) 2537.1536 2714.3464
TreatmentB   133.0911  244.3534
```

This method does not estimate confidence intervals for the parameters of the random effects so the corresponding elements are returned as “NA” meaning “Not Available”.

Confidence intervals for the model parameters can also be estimated through bootstrap methods as in the following example.

```
> # create cluster
> cl <- makeCluster(spec=rep("localhost", 2), type="PSOCK");
>
> # set RNG seed on cluster
> clusterSetRNGStream(cl=cl, iseed=iseed);
>
> # parametric bootstrap confidence intervals
> confint(
+   fit0,
+   level=0.95,
+   method="boot",
+   nsim=10000,
+   boot.type="perc",
+   type="parametric",
+   cl=cl
+ );
Computing bootstrap confidence intervals ...
Warning messages:
1: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,  :
  unable to evaluate scaled gradient
2: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,  :
  Model failed to converge: degenerate  Hessian with 1 negative eigenvalues
                 2.5 %     97.5 %
.sig01          0.00000   79.00614
.sig02          0.00000   80.12621
.sig03         64.05394  202.76604
.sigma         76.32105  126.74377
(Intercept) 2536.58251 2714.83184
TreatmentB   133.28439  244.03765
Warning messages:
1: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,  :
  unable to evaluate scaled gradient
2: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv,  :
  Model failed to converge: degenerate  Hessian with 1 negative eigenvalues
>
> # stop cluster
> stopCluster(cl=cl);
```

In this example, a cluster of two R sessions for parallel computation is opened on the local machine. The seed for the random number generator is set on the parallel cluster. Afterwards the “confint” function is called to compute the confidence interval of the model parameters by bootstrap using parallel computation. Finally, the cluster of parallel R sessions is closed. Now let us look more carefully to the arguments of the “confint” function. By supplying the “boot” string to the “method” argument a bootstrap method is defined. The “nsim” argument accepts integer values defining the number of bootstrap samples. To increase the precision of estimation this value should be increased. A choice of 10000 samples leads to reasonably precise estimates but the computation can be time consuming. The percentile is used to find the lower and upper bounds of the confidence interval as defined in the

"boot.type" argument. The "type" argument defines the type of bootstrap, which is a parametric bootstrap in this example. The "cl" argument defines the cluster of R sessions for parallel computation. Bootstrap methods can take a very long time to compute. Parallel computation can significantly decrease the time needed to complete a computation depending on the number of cores present on the local machine and on the frequency of each core.

7.2 Estimation and inference on individual treatment effects

Linear mixed-effects models can be used to obtain an overall treatment effect as well as individual treatment effects [10, 11]. The estimates of the individual treatment effects obtained in this way are generally more precise than estimates obtained through equivalent fixed effect only linear models. Individual treatment effect estimates obtained through linear mixed-effect modelling are closer to the overall treatment effect than estimates obtained through fixed effect only linear models, and that's why individual linear mixed-effect model estimates are referred to as shrunk estimates in the statistical literature [12]. An individual treatment effect is defined as the expected difference of outcome variable registered under two treatments for a given individual while keeping additional variables constant. As such for linear mixed-effects models, individual treatment effects depend only on the fixed effects and on random effects that contain both the patient and the treatment categorical variables simultaneously.

Before computing the individual treatment effects, a data.frame containing all the variables used to fit the linear mixed-effects model must be defined. This data.frame must contain the values of the "Patient" and "Treatment" factors for which predictions are desired.

```
> # prediction data.frame
> pred_newdata0 <- expand.grid(
+   unique(ndata$Treatment),
+   unique(ndata$Patient)
+ );
```

The "pred_newdata0" data.frame that has just been created contains two rows for each patient, one row for each treatment. Since the individual treatment effects do not depend on variables other than the "Patient" and "Treatment", any additional variable used to fit the linear mixed-effects model must

be arbitrarily kept constant at any of the values it can assume. In this case, the “Cycle” factor is set to the reference level for any patient and treatment.

```
> # add 'Cycle' factor to data.frame
> pred_newdata0 <- cbind(
+   pred_newdata0,
+   rep(
+     x=levels(ndata$Cycle)[1], # the reference level
+     times=nrow(pred_newdata0)
+   ) # note that additional variables are kept constant
+ );
```

These lines of code essentially add a column to the “pred_newdata0” data.frame and define all of the elements in this column to the reference level of the “Cycle” factor. To finalize the definition of the prediction data.frame, its variables must be named correctly.

```
> # name variables in prediction data.frame
> names(pred_newdata0) <- names(ndata)[c(2, 1, 3)];
```

Here the names of the columns of “pred_newdata0” are set to the names of columns number 2, 1 and 3 of the dataset used to fit the linear mixed-effects model. To check that the prediction data.frame has been correctly defined let us print the first six rows.

```
> # print first 6 rows of data.frame
> head(pred_newdata0, n=6);
  Treatment Patient Cycle
1         A       1     1
2         B       1     1
3         A       2     1
4         B       2     1
5         A       3     1
6         B       3     1
```

To get the individual treatment effects first the predicted values for each patient and treatment must be obtained.

```
> # compute predicted values for each patient and treatment
> pred0 <- predict(
+   object=fit0,
+   newdata=pred_newdata0,
+   re.form=~(1|Patient)+(1|Patient:Treatment)
+ );
```

The data contained in the “fit0” object resulting from a previous “lmer” fit is used as argument. The predicted values are computed for the values of the variables defined within the “pred_newdata0” data.frame supplied to the “newdata” argument. The “re.form” argument instructs the “predict”

function to condition on the patient and treatment when computing the predicted values. The predicted values are saved under the “pred0” object. To obtain the individual treatment effects one must take a predicted value for each patient under treatment “B” and subtract the predicted value for treatment “A” computed under the same patient and cycle. The next lines of code do just that.

```
> # compute individual treatment effects
> n <- nlevels(pred_newdata0$Patient); # number of patients
> pdata0 <- data.frame(
+   "Patient"=numeric(n),
+   "Effect"=numeric(n)
+ );
> index <- 1;
> for ( patient in levels(pred_newdata0$Patient) ) { # loop through the patients
+   indexB <- with(pred_newdata0,
+                  which(Patient==patient & Treatment=="B")
+   );
+   indexA <- with(pred_newdata0,
+                  which(Patient==patient & Treatment=="A")
+   );
+   pdata0$Patient[index] <- patient;
+   pdata0$Effect[index] <- pred0[indexB]-pred0[indexA];
+   index <- index+1; # increase index for next loop
+ }
> pdata0$Patient <- factor(pdata0$Patient);
>
> # reorder factor levels on data.frame
> pdata0$Patient <- factor(
+   x=pdata0$Patient,
+   levels=levels(pdata0$Patient)[
+     order( as.numeric( levels(pdata0$Patient) ) )
+     ]
+ );
>
> print(pdata0); # print results
   Patient    Effect
1        1 195.1297
2        2 169.6425
3        3 165.1196
4        4 217.9276
5        5 201.6696
6        6 163.2860
7        7 186.2061
8        8 182.2944
9        9 213.5880
10      10 199.5303
11      11 193.4183
12      12 176.8547
```

The individual treatment effects are saved to a data.frame object named “pdata0” where the respective patient can be identified. Note that these individual treatment effect estimates are closer to the overall treatment effect of 188.7222 units than the naïve individual treatment effect estimates presented above.

```
> # range of shrunk estimates
> range(pdata0$Effect);
[1] 163.2860 217.9276
>
> # range of naive estimates
> range(sddata);
[1]  50 348
```

The shrunk individual treatment effect estimates range between around 163 and 217 units, while the naïve individual treatment effect estimates range between 50 and 348 units. Thus, the shrunk individual treatment effect estimates exhibit lower variance across all the patients than their naïve counterparts.

Parametric bootstrap estimation of the standard error and confidence interval of each individual treatment effect is currently under research. The results of this research will be presented in later publications. Instead individual treatment effect standard errors and confidence intervals can be obtained using ready available random effect meta-analysis methodology [13]. Package “metafor” is required for the next examples and must be installed first [14].

```
> # install package if not installed
> if ( !"metafor" %in% installed.packages() ) {
+   install.packages("metafor");
+ }
Installing package into ‘C:/Users/araujo/Documents/R/win-library/3.2’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.2/metafor_1.9-8.zip'
Content type 'application/zip' length 2175656 bytes (2.1 MB)
downloaded 2.1 MB

package ‘metafor’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\araujo\AppData\Local\Temp\Rtmp2xhYMy\downloaded_packages
```

After “metafor” is installed it must be loaded into the R environment.

```
> library(metafor);
Loading 'metafor' package (version 1.9-8). For an overview
and introduction to the package please type: help(metafor).
```

Next, a random effect meta-analysis model is fitted to the dataset where the outcome variable is recoded as the difference of outcome variable registered under the two treatments administered to the patients on the same cycle. Remember that this dataset is saved under the name “ddata”. A linear mixed effects model of differences can be fitted to this dataset. The variance of the residual error of the linear mixed-effect model of differences is twice the variance of the residual error in the equivalent full linear mixed-effects model. This implies that the standard deviation of the residual error in the latter model must be multiplied by the square root of two to obtain the standard deviation of the residual error in the former model.

```
> # compute standard deviation of residual error
> sigma0 <- sqrt(2)*sigma(fit0);
```

By assuming the same standard deviation for all the patients, the random effect meta-analysis behaves like a linear mixed-effect model yielding the exact same individual treatment effects as can be seen next. Therefore, the following lines of code compute the individual means of outcome difference and their respective standard errors assuming homoscedasticity, i.e. constant variance of residual error.

```
> # compute per patient mean of differences
> n <- nlevels(ddata$Patient);
> mdata <- data.frame(
+   "Patient"=numeric(n),
+   "k"=numeric(n),
+   "mean"=numeric(n),
+   "se"=numeric(n)
+ );
> index <- 1;
> for ( patient in levels(ddata$Patient) ) { # loop through the patients
+   sub <- ddata[ddata$Patient==patient,];
+   mdata$Patient[index] <- patient;
+   mdata$k[index] <- nrow(sub);
+   mdata$mean[index] <- mean(sub$dY);
+   mdata$se[index] <- sigma0/sqrt( nrow(sub) );
+   index <- index+1; # increase index for next loop
+ }
> print(mdata);
   Patient k      mean       se
1        1 3 223.66667 88.85371
2        2 3  84.66667 88.85371
3        3 3  60.00000 88.85371
4        4 3 348.00000 88.85371
5        5 3 259.33333 88.85371
6        6 3  50.00000 88.85371
7        7 3 175.00000 88.85371
8        8 3 153.66667 88.85371
9        9 3 324.33333 88.85371
10      10 3 247.66667 88.85371
11      11 3 214.33333 88.85371
12      12 3 124.00000 88.85371
```

The resulting data are saved on the “mdata” data.frame, which contains four variables; a “Patient” variable identifying the patient, a “k” variable holding the number of cycles, a “mean” variable containing the naïve individual treatment effect and finally a “se” variable holding its respective standard error. The “mdata” data.frame contains all the necessary information for the random effect meta-analytic model fit that follows.

```
> # fit random effect meta-analysis model
> orma <- rma.uni(
+   yi=mean,
+   sei=se,
+   weights=k/sum(k),
+   data=mdata,
+   method="REML"
+ );
```

The “rma.uni” function is used to fit a meta-analysis model through the restricted maximum-likelihood estimator as defined in the “method” argument. The “data” argument instructs the function to look

into the “mdata” data.frame for the variables. The “yi” argument defines the individual means and the “sei” argument their respective standard error. To yield results that are equivalent to a linear mixed-effects model analysis the individual means must be weighted by their respective standard errors and number of observations. The “weights” argument correctly defines the individual weights as the ratio of the respective number of observations to the total number of observations. The random effect meta-analytic model is saved under the “orma” object. Finally, the individual treatment effects, their standard errors and confidence intervals can be computed.

```
> # the individual treatment effects
> blup(orma, level=0.95);
       pred      se    pi.lb    pi.ub
1  195.1297 44.5523 107.8087 282.4507
2  169.6425 44.5523  82.3215 256.9635
3  165.1196 44.5523  77.7986 252.4406
4  217.9276 44.5523 130.6066 305.2486
5  201.6696 44.5523 114.3486 288.9905
6  163.2860 44.5523  75.9650 250.6070
7  186.2061 44.5523  98.8851 273.5271
8  182.2944 44.5523  94.9734 269.6154
9  213.5880 44.5523 126.2670 300.9090
10 199.5303 44.5523 112.2093 286.8513
11 193.4183 44.5523 106.0973 280.7393
12 176.8547 44.5523  89.5337 264.1757
```

The random effect meta-analytic “orma” object is supplied to the “blup” function. Here the “level” argument defines a 95% coverage for the confidence interval. Note that the same individual treatment effects are obtained as in the linear mixed-effect model displayed in the print of the “pdata” object. Also, note that the estimated standard error of the individual treatment effects is about half the standard error of the naïve individual treatment effects displayed in the print of the “mdata” data.frame. This fact suggests an increased precision of the shrunk estimates relative to the naïve ones. Finally, it can also be observed that none of the confidence intervals of the individual treatment effects contains zero, suggesting that the treatments are significantly different for each patient at the usual 5% significance level.

8 Linear mixed-effects model of difference

A simpler linear mixed-effects model can be fitted to the smaller recoded dataset presented above. This model called linear mixed-effects model of difference, can be obtained from the linear mixed-effects model by differencing the outcome variable indexed to one treatment and the other on the same cycle and patient. Note that the outcome variable in the recoded "ddata" dataset is obtained from the full "ndata" dataset by an identical process. In this computation process, the patient and cycle random effects are eliminated out of the equation and the treatment by patient and residual random effects remain. In the linear mixed effects model of difference there is an effect indexed to the patient that is equivalent to the difference of a random treatment by patient interaction indexed on the same patient and cycle in the linear mixed-effects model. While the residual error of the same linear mixed-effects model of difference equals the difference of the residual errors indexed on the same patient and cycle from the linear mixed-effects model. Given that the random effects of the linear mixed-effects model are independent by assumption, the variance of the patient effect in the linear mixed-effects model of difference equals twice the variance of the random treatment by patient interaction effect of the equivalent linear mixed-effect model. While the variance of the residual error of the former model equals twice the variance of the residual error in the latter model. The usage of the functions present in the following examples is identical to the usage described above when the linear mixed effects model is fitted to the full dataset; as such, details of each of the arguments are skipped. So one starts by fitting a linear mixed-effects model to the recoded "ddata" dataset using restricted maximum-likelihood for estimation.

```
> ## Mixed model using differences ##
> fit1 <- lmer(formula=dY~1+(1|Patient), data=ddata, REML=TRUE);
> print(fit1);
Linear mixed model fit by REML ['lmerMod']
Formula: dY ~ 1 + (1 | Patient)
   Data: ddata
REML criterion at convergence: 457.6782
Random effects:
 Groups   Name        Std.Dev.
 Patient  (Intercept)  42.1
 Residual              153.9
Number of obs: 36, groups:  Patient, 12
Fixed Effects:
(Intercept)
      188.7
```

8.1 Estimation and inference on the overall treatment effect

In this model, there is only one fixed effect, the intercept, which is equal to the overall treatment effect. Also, note that the random effects part of the model contains a patient random term and the residual error.

```
> # extract the fixed effects
> fixef(fit1);
(Intercept)
   188.7222
```

The variance and standard error of the random effects terms of the model can be obtained as above.

```
> vfit1 <- VarCorr(fit1);
> print( x=vfit1, comp=c("Variance", "Std.Dev") );
 Groups    Name        Variance Std.Dev.
 Patient   (Intercept)  1772.7   42.103
 Residual               23684.9  153.899
```

The reader can now check the mentioned relationship between the variance of the random terms of the linear mixed-effects model and the linear mixed-effects model of difference. The variance components of the linear mixed-effects model are printed again to ease the comparison.

```
> print( x=vfit0, comp=c("Variance", "Std.Dev") );
 Groups             Name        Variance Std.Dev.
 Patient:Cycle      (Intercept)   680.53  26.087
 Patient:Treatment  (Intercept)   886.34  29.771
 Patient            (Intercept) 19459.14 139.496
 Residual                       11842.47 108.823
```

In this case, the covariance matrix of the fixed effects contains a single element, the estimate of the variance of the overall treatment effect.

```
> # covariance matrix of the fixed effects
> vcov(fit1);
1 x 1 Matrix of class "dpoMatrix"
             (Intercept)
(Intercept)    805.6378
```

The Kenward-Roger approximation of the covariance matrix of the fixed effects can be obtained as well.

```
> vcovAdj(fit1);
1 x 1 Matrix of class "dgeMatrix"
             (Intercept)
(Intercept)     805.6378
```

Since the “lmer” function cannot fit models without fixed effects, the likelihood ratio test cannot be used to test the significance of the intercept term of the linear mixed-effects model of difference. Fitting a reduced model without the intercept would leave no fixed effects in the model. In this case, function “lmer” always fits an intercept as fixed no matter what is tried. However, the confidence interval of the intercept can be used to test its significance as is explained below. Parametric bootstrap is used to estimate the confidence intervals of the parameters of the model.

```
> # create cluster
> cl <- makeCluster(spec=rep("localhost", 2), type="PSOCK");
>
> # set RNG seed on cluster
> clusterSetRNGStream(cl=cl, iseed=iseed);
>
> # parametric bootstrap confidence intervals
> confint(
+   fit1,
+   level=0.95,
+   method="boot",
+   nsim=10000,
+   boot.type="perc",
+   type="parametric",
+   cl=cl
+ );
Computing bootstrap confidence intervals ...
                2.5 %   97.5 %
.sig01         0.0000 109.0464
.sigma       109.6905 188.7785
(Intercept) 131.5533 243.9363
Warning messages:
1: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
2: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
3: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
4: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
5: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
6: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
7: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
8: In optwrap(object@optinfo$optimizer, ff, x0, lower = lower, control = control$optCtrl,  :
  convergence code 3 from bobyqa: bobyqa -- a trust region step failed to reduce q
>
> # stop cluster
> stopCluster(cl=cl);
```

Note that the 95% confidence interval of the intercept does not contain zero, suggesting that the overall treatment effect is significantly different from zero at the 5% significance level. There is statistical evidence that the treatments differ at a 5% significance level.

8.2 Estimation and inference on individual treatment effects

For the linear mixed-effects model of difference, the individual treatment effects can be obtained directly from the predicted values. To start, a data.frame with all the levels of the "Patient" factor is created.

```
> # prediction data.frame
> pred_newdata1 <- data.frame(
+   "Patient"=unique(ddata$Patient)
+ );
```

Now the predicted values for each individual can be obtained from the "fit1" object resulting from the linear mixed-effects model fit to the dataset of outcome differences, and the "pred_newdata1" dataset holding the values of the variables for which predictions are to be evaluated.

```
> # compute predicted values for each patient
> pred1 <- predict(
+   object=fit1,
+   newdata=pred_newdata1,
+   re.form=~(1|Patient)
+ );
> pdata1 <- data.frame(pred_newdata1, "Effect"=pred1);
> print(pdata1);
   Patient    Effect
1        1 195.1297
2        2 169.6425
3        3 165.1196
4        4 217.9276
5        5 201.6696
6        6 163.2860
7        7 186.2061
8        8 182.2944
9        9 213.5880
10      10 199.5303
11      11 193.4183
12      12 176.8547
```

The individual treatment effects obtained through the linear mixed-effects model of difference match the values resulting from the linear mixed-effects model and the random effects meta-analysis approach devised to mimic a linear mixed-effects model.

The code written to obtain the standard errors and confidence intervals of the individual treatment effects through the random effects meta-analysis is very similar to the one presented above in this document. The only difference is that now the standard deviation of the residual error can be directly obtained from the linear mixed effects model of difference object.

```
> # compute standard deviation of residual error
> sigma0 <- sigma(fit1);
>
> # compute per patient mean of differences
> n <- nlevels(ddata$Patient);
> mdata <- data.frame(
+   "Patient"=numeric(n),
+   "k"=numeric(n),
+   "mean"=numeric(n),
+   "se"=numeric(n)
+ );
> index <- 1;
> for ( patient in levels(ddata$Patient) ) { # loop through the patients
+   sub <- ddata[ddata$Patient==patient,];
+   mdata$Patient[index] <- patient;
+   mdata$k[index] <- nrow(sub);
+   mdata$mean[index] <- mean(sub$dY);
+   mdata$se[index] <- sigma0/sqrt( nrow(sub) );
+   index <- index+1; # increase index for next loop
+ }
> orma <- rma.uni(
+   yi=mean,
+   sei=se,
+   weights=k/sum(k),
+   data=mdata,
+   method="REML"
+ );
> blup(orma, level=0.95); # the individual treatment effects
       pred      se    pi.lb    pi.ub
1  195.1297 44.5523 107.8087 282.4507
2  169.6425 44.5523  82.3215 256.9635
3  165.1196 44.5523  77.7986 252.4406
4  217.9276 44.5523 130.6066 305.2486
5  201.6696 44.5523 114.3486 288.9906
6  163.2860 44.5523  75.9650 250.6070
7  186.2061 44.5523  98.8851 273.5271
8  182.2944 44.5523  94.9734 269.6154
9  213.5880 44.5523 126.2670 300.9090
10 199.5303 44.5523 112.2093 286.8513
11 193.4183 44.5523 106.0973 280.7393
12 176.8547 44.5523  89.5337 264.1757
```

The exact same results as for the linear mixed-effects model are obtained. The conclusions are obviously the same and are not repeated here.

9 Graphical methods

Graphical representations provide means to visualize and interpret the data that are far easier to understand when compared to some of the statistical methods presented above. In this chapter, graphical representations of data arising from n-of-1 trials are demonstrated. As mentioned several times above in this document, n-of-1 trials are run when there is suspicion that the subjects respond distinctively to treatments, so that a treatment might be recommended for some patients but not for others. In the medical statistical parlance, this phenomenon is referred to as treatment by patient interaction. Trellis plots are graphical representations that display a variable or the relationship between variables, conditioned on one or more variables. Therefore, trellis plots are a particularly useful tool for displaying n-of-1 trial data, where the relationship between an outcome variable and the treatment categorical variable can be represented conditioned on the subject categorical variable. Such plots allow for an observation of the magnitude of the difference of the outcome variable for each of the treatments studied within and between the subjects recruited into the trial. This advantage of trellis plots is more evident below where the actual plots are presented. The examples of trellis plots demonstrated here are only a fraction of what can be done with trellis plots within R. For plots not foreseen here and advanced trellis plotting, the consultation of the work of Sarkar [15] is suggested.

Before running the code that draws the trellis plots, it is necessary to load the “lattice” package that is supplied with the base R distribution for some operating systems.

```
> # load lattice package
> library(lattice);
```

9.1 Scatterplots

The demonstration starts with the plot that is more difficult to code. This way there is an opportunity to explain how trellis plotting works with the “lattice” package within the R software. As an example, consider a scatterplot of the outcome under both treatments for each patient based on the dataset of differences previously presented. Remember that there are twelve patients in this dataset, so that twelve Cartesian plots have to be drawn side by side. To accomplish this the developer of package “lattice” devised a way that is not that difficult for R programmers! The trellis subplots are called panels within R. Therefore; twelve panels are drawn in this case. Package “lattice” provides several panel functions that R programmers can use to draw plots on each panel. The following lines of code define a panel function that is saved for subsequent use.

```
> # define panel function
> mypanel_00 <- function(
+     x,
+     y,
+     subscripts,
+     groups,
+     ...
+ ) {
+     panel.xyplot(
+         x,
+         y,
+         subscripts=subscripts,
+         groups=groups,
+         type="p", # type of plot 'p' for points
+         cex=1, # character size
+         pch=16, # character 16
+         col="blue", # color 'blue'
+         ...
+     );
+     panel.abline(
+         a=0, # intercept
+         b=1, # slope
+         lty="solid", # line type 'solid'
+         lwd=1, # line width
+         col="black" # color 'black'
+     );
+     panel.points(
+         x=tapply(X=x, INDEX=groups[subscripts], FUN=mean),
+         y=tapply(X=y, INDEX=groups[subscripts], FUN=mean),
+         cex=1, # character size
+         pch=8, # character 8
+         col="red", # color 'red'
+         ...
+     );
+ } # mypanel_00
```

The above function has four mandatory arguments, i.e. arguments that must be supplied when calling the function. Argument “x” is a vector of x-axis coordinates, and argument “y” is a vector of corresponding y-axis coordinates. The “x” and “y” vectors are required to plot points on each panel according to the Cartesian coordinate system. The “subscripts” argument defines the indexes of the two previous vectors that are actually plotted. Moreover, the “groups” argument can be any categorical variable. The set of arguments finishes with three dots “…” meaning that additional arguments can be supplied. These additional arguments are passed to the functions called inside “mypanel_00”. Within “mypanel_00”, three panel functions are called. Under “lattice” package, “xyplot” is used to plot scatterplots. The default panel function for “xyplot” is named “panel.xyplot” and it is the first one called inside function “mypanel_00”. It essentially plots points on each panel

defined by the levels of a conditioning factor. The second function call “panel.abline”, is used to draw the line of equality, which is defined by a zero intercept and a slope equal to unity. The last panel function being called is “panel.points”, and it is used here to plot the mean point of the points previously drawn by “panel.xyplot”. Arguments of the three panel functions have comments on the same line when appropriate. The purpose of these comments is to inform the reader as regards the meaning of the corresponding argument. Consequently, there is no need to explain each argument here. Also, note the three dots “…” are an argument to each panel function. This ensures that any additional arguments supplied to “mypanel_00” function are effectively passed to each panel function called inside it.

A graphical legend under the “lattice” language is called a “key”. A list of parameters required to draw the legend on the plot can be supplied to a so-called “key” parameter. Therefore, the next step is to define and save the “key” parameters.

```
> # define legend
> mykey_00 <- list(
+   space="top", # put legend at the top of the plot area
+   points=list(cex=1, pch=16, col="blue"),
+   text=list("FEV(ml) by cycle"),
+   lines=list(lty="solid", lwd=1, col="black"),
+   text=list("Line of equality"),
+   points=list(cex=1, pch=8, col="red"),
+   text=list("FEV(ml) mean")
+ );
```

The list of parameters for the legend is saved under the object named “mykey_00”. The name of each element of the list is equal to the name of the parameter and the corresponding list element stores the actual parameter definition. The parameter definition can be another list of parameters as is the case here for the “points”, “lines” and again “points” parameters. The legend is to be placed at the top of the plot area as specified by the “space” parameter. The “points”, “text”, “lines”, “text”, “points”, “text” parameters are specified to draw the specified legend elements in columns in that order.

In the lines of code that follow, the “xyplot” function is called and the object it returns is saved under the name “trellis_00”. Lattice plot functions return an object of class “trellis”, so “trellis_00” is an object of this class.

```
> # save trellis data
> trellis_00 <- xyplot(
+   YB~YA|Patient, # plot YB vs YA for each level of the 'Patient' factor
+   data=ddata, # dataset
+   groups=Patient, # 'Patient' as groups needed to plot the mean
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list( c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3) ),
+   panel=function(x, y, subscripts, groups, ...) {
+     mypanel_00(x, y, subscripts, groups, ...);
+   }, # panel function
+   xlab="Treatment A", # x axis label
+   ylab="Treatment B", # y axis label
+   aspect="iso", # aspect ratio same scale on both axis
+   key=mykey_00 # list of legend parameters
+ );
```

Here the first argument is a formula that instructs “xyplot” to construct scatterplots of outcome under treatment “B” in the ordinates axis against outcome under treatment “A” in the abscissa axis for each level of the “Patient” factor. The variables in the formula are looked for in the dataset supplied to the “data” argument, in this case “ddata”. The “groups” argument is set to the variable “Patient” in the dataset. This argument is to be passed to the “mypanel_00” function defined above, and it is needed to plot the mean point for each panel. The “layout” parameter instructs “xyplot” to distribute the panels as in a table with three columns and four rows. The “index.cond” parameter defines the assignment of the levels of the conditioning variable to the panels on the plot. This parameter is specified because plotting functions in “lattice” package draw panels from left to right and from bottom to top, but the desired order of the panels is distinct. Now it is important to note the “panel” argument, which is set to a function that calls the “mypanel_00” function saved earlier. This function is used to plot the data on each of the panels defined by the levels of the conditioning factor. The “xlab” and “ylab” arguments define the labels of the x-axis and y-axis respectively. By setting “aspect” to “iso”, the same scale is drawn for each of the two axis of the Cartesian plot drawn for each panel. This is useful when the dimensions of the variables drawn on both axis are the same, like in this particular case. Finally, the list of legend parameters saved under the “mykey_00” object is passed to the “key” parameter, instructing “xyplot” to draw a legend according to those parameters.

To effectively plot the data and produce the desired graph the “trellis_00” object returned by “xyplot” must be printed.

```
> # plot trellis scatterplot
> print(trellis_00);
```

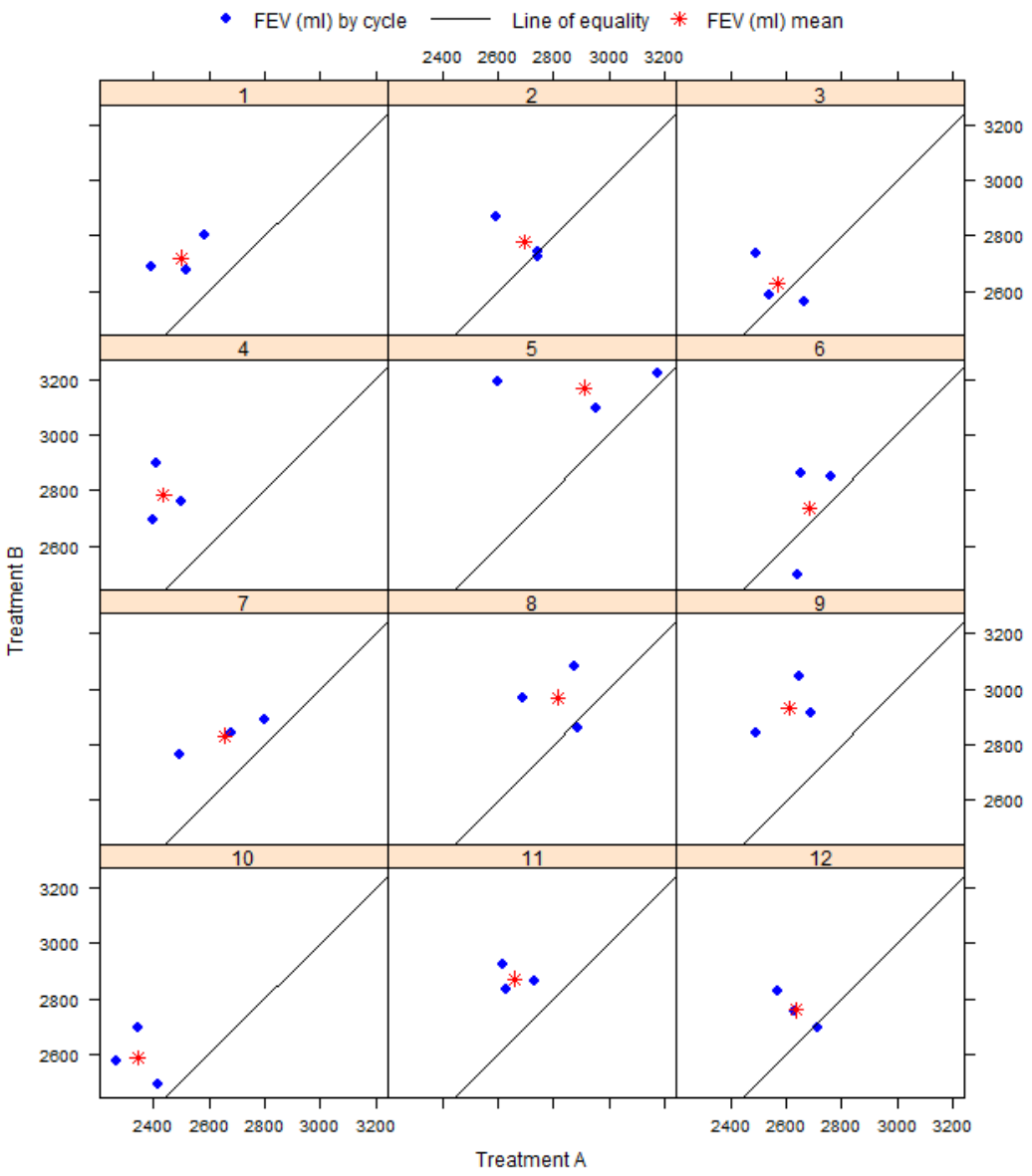


Figure 9.1: Outcome under treatment "B" versus outcome under treatment "A" for each patient.

After running the print of the saved "trellis_00" object, the graph presented in Figure 9.1 is created. Note that the points and lines displayed under each panel are drawn by a call to the "mypanel_00" function defined above. Remember that calls to "panel.xyplot", "panel.abline", and "panel.points" are made inside "mypanel_00". Also note that the blue points in Figure 9.1 are drawn by a call to "panel.xyplot"; the solid black lines are drawn by a call to "panel.abline"; and the red star like points are drawn by a call to "panel.points". As regards the interpretation of the plot, note that for each patient there is a point for each cycle (in blue) and that these points are scattered around the mean point (in red) which is the central point. The solid black line is the line of equality of outcome under two treatments. Points falling above the line and closer to the axis labelled "Treatment B" are points for which the outcome is superior under treatment "B" than under treatment "A". Points falling below

the equality line and closer to the axis labelled treatment “A” are points for which the reverse happens. For patients number 3, 6, 8 and 12 there is at least a point that falls below the line, but the majority of the cycle points and the mean point fall above the equality line for all the twelve patients without exception. Hence, the graph suggests that the outcome is superior on average under treatment “B” for all the patients studied. Therefore, if it is considered that the health condition under study improves as the outcome variable registers higher values, treatment “B” is preferable over treatment “A”. If it is the other way around, than the choice shall fall over treatment “A”.

An even more interesting plot is the one presented in the next example. On this example, the shrunk estimates of the individual treatment effects and the estimates of the overall treatment effect are plotted alongside the cycle observations and the naïve individual treatment effect estimates. Estimation of shrunk individual treatment effects through linear mixed-effects modelling is presented on chapters 7, and 8 above. To start, a vector with the shrunk predictions for each observation is computed from object “fit0” and saved for later use.

```
> # shrunk predictions for all observations
> pred_01 <- predict(
+   fit0,
+   re.form=~(1|Patient)+(1|Patient:Treatment)
+ );
```

Afterwards a list with the Cartesian coordinates of the shrunk estimates is computed from the “pred_01” object that has just been saved. This list of coordinates is later used to plot points of the shrunk mean for each patient studied.

```
> # list with shrunk coordinates for plotting
> shrunk_01 <- list(
+   "x"=pred_01[
+     model.frame(fit0)$Treatment==levels(model.frame(fit0)$Treatment)[1]
+     ],
+   "y"=pred_01[
+     model.frame(fit0)$Treatment==levels(model.frame(fit0)$Treatment)[2]
+     ]
+ );
```

To plot the overall mean point on each panel, a similar list of coordinates must be computed and saved for posterior use.

```
> overall_01 <- list(
+   "x"=fixef(fit0)[1], # coordinate under treatment 'A'
+   "y"=fixef(fit0)[1]+fixef(fit0)[2] # coordinate under treatment 'B'
+ );
> names(overall_01$x) <- names(overall_01$y) <- NULL;
```

The coordinates of the overall mean are computed from the “fit0” object that is presented on chapter 7. Note that the abscissa is equal to the fixed effects intercept in the linear mixed-effects model. Also, note that the ordinate is equal to the intercept added to the coefficient of the “Treatment” factor used in the fixed effects part of the same model.

The next step is to define a panel function to be used later on the “xyplot” call.

```
> # define panel function
> mypanel_01 <- function(
+   x,
+   y,
+   z, # list of shrunk mean coordinates
+   o, # list of overall mean coordinates
+   subscripts,
+   groups,
+   ...
+ ) {
+   panel.xyplot(
+     x,
+     y,
+     subscripts=subscripts,
+     groups=groups,
+     type="p", # type of plot 'p' for points
+     cex=1, # character size
+     pch=16, # character 16
+     col="blue", # color 'blue'
+     ...
+   ); # plot cycle points
+   panel.abline(
+     a=0, # intercept
+     b=1, # slope
+     lty="solid", # line type 'solid'
+     lwd=1, # line width
+     col="black" # color 'black'
+   ); # plot line of equality
+   panel.points(
+     x=tapply(X=x, INDEX=groups[subscripts], FUN=mean),
+     y=tapply(X=y, INDEX=groups[subscripts], FUN=mean),
+     cex=1, # character size
+     pch=8, # character 8
+     col="red", # color 'red'
+     ...
+   ); # plot naive mean
+   panel.points(
+     x=z$x[subscripts][1],
+     y=z$y[subscripts][1],
+     cex=1, # character size
+     pch=4, # character 4
+     col="green", # color 'green'
+     ...
```

```
+   ); # plot shrunk mean
+   panel.points(
+     x=o$x,
+     y=o$y,
+     cex=1, # character size
+     pch=3, # character 3
+     col="dark gray", # color 'dark gray'
+     ...
+   ); # plot overall mean
+ } # mypanel_01
```

The “mypanel_01” function is essentially based on the “mypanel_00” function used to plot Figure 9.1. Two arguments are added (“z” and “o”) to pass the coordinates of the shrunk means for each patient and the coordinates of the overall mean to the inside of the function. In addition, two calls to “panel.points” are added to plot the patient shrunk means and the overall mean on each panel.

To identify the additional points an appropriate legend must be defined.

```
> # define legend
> mykey_01 <- list(
+   rep=FALSE,
+   space="top", # put legend at the top of the plot area
+   points=list(cex=1, pch=16, col="blue"),
+   text=list(labels="FEV(ml) by cycle", cex=1),
+   lines=list(lty="solid", lwd=1, col="black"),
+   text=list(labels="Line of equality", cex=1),
+   points=list(
+     cex=1,
+     pch=c(8, 4, 3),
+     col=c("red", "green", "dark gray")
+   ),
+   text=list(
+     labels=c("Naive mean", "Shrunk mean", "Overall mean"),
+     cex=1
+   )
+ );
```

Note that in relation to the previous plot, the legend is rewritten to accommodate for the two additional points. The legend labels are also laid out in three columns at the top of the plot area. The first and second columns contain only one row. The third and last column contain three rows one for each mean point drawn. By default rows not specified in the list of parameters are repeated a number of times equal to the maximum number of rows present in the list. The “rep=FALSE” argument avoids this unnecessary repetition.

Finally, the data for the trellis plot can be computed.

```
> # save and print trellis data
> trellis_01 <- xyplot(
+   YB~YA|Patient, # plot YB vs YA for each level of the 'Patient' factor
+   data=ddata, # dataset
+   groups=Patient, # 'Patient' as groups needed to plot the mean
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list( c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3) ),
+   panel=function(
+     x,
+     y,
+     subscripts,
+     groups,
+     z=shrunk_01,
+     o=overall_01,
+     ...
+   ) {
+     mypanel_01(x, y, z, o, subscripts, groups, ...);
+   }, # panel function
+   xlab="Treatment A", # x axis label
+   ylab="Treatment B", # y axis label
+   aspect="iso", # aspect ratio same scale on both axis
+   key=mykey_01 # list of legend parameters
+ );
> print(trellis_01);
```

The “xyplot” call is similar to the one leading to the plot of Figure 9.1 with two obvious differences. The panel function calls “mypanel_01” instead of “mypanel_00”. In addition, the list of parameters supplied to the “key” parameter is now “mykey_01” instead of “mykey_00”. Also note that the “shrunk_01” object holding a list of coordinates of the shrunk estimates, and the “overall_01” object holding a list of overall mean coordinates, are passed to the panel function within this last “xyplot” call. A “trellis” object is returned from the “xyplot” call, and saved under the name “trellis_01”. The print of the “trellis_01” object leads to the plot of Figure 9.2.

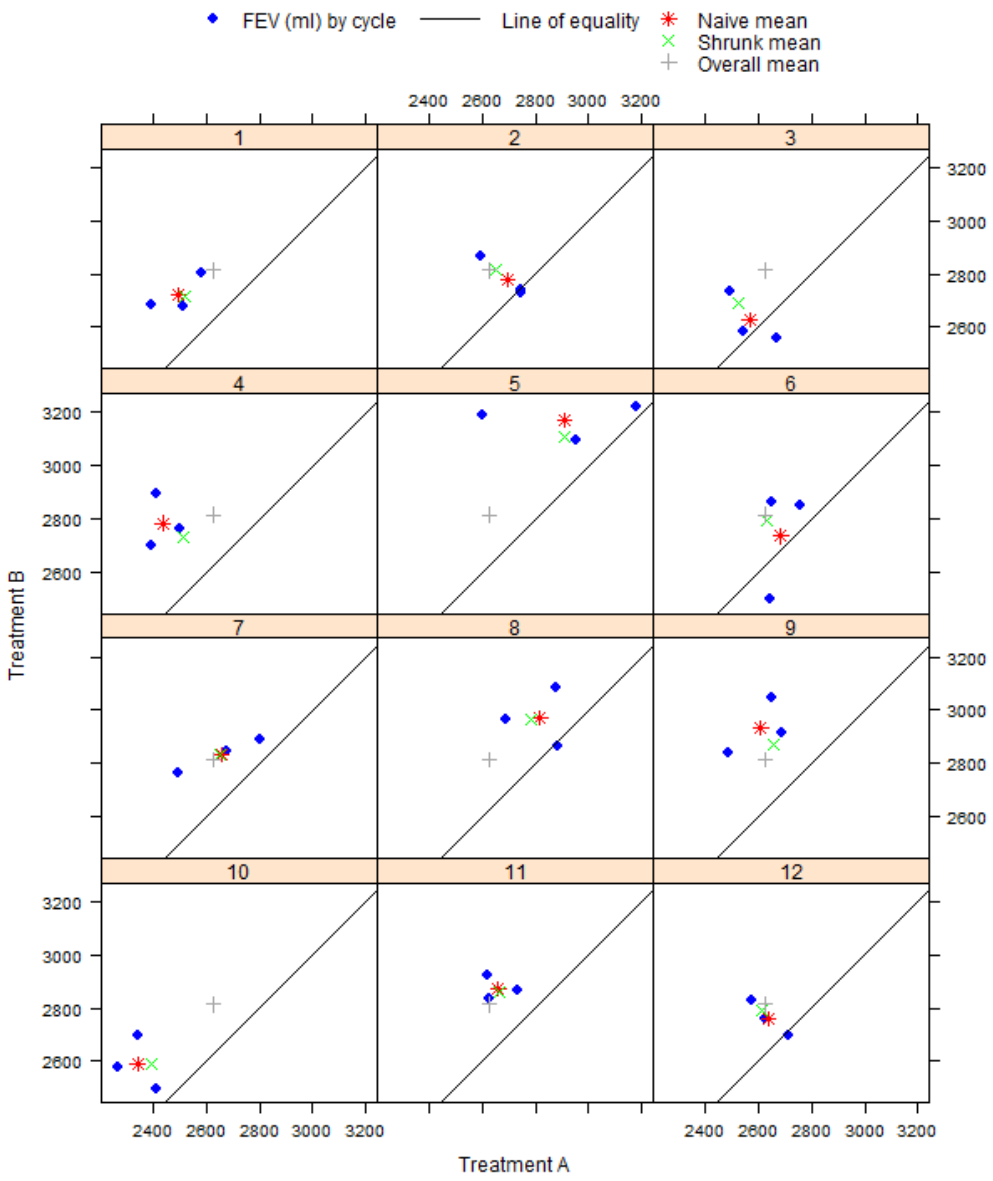


Figure 9.2: Outcome under treatment "B" versus outcome under treatment "A" for each patient.

The two additional points do not change the interpretations made above in respect to Figure 9.1. The shrunk mean and the overall mean remain above the line of equality suggesting that higher values of outcome variable are observed under treatment "B". The overall mean is an estimate of the treatment effect across all the patients studied. It is therefore the same for all the patients. The data is balanced, i.e. the patients received treatments in the same number of cycles, so the overall mean resulting from the linear mixed-effects model presented on chapter 7 is the same as the naïve overall mean. The shrunk individual treatment effect or shrunk mean is a weighted average of the overall mean and the naïve individual mean, where the weights depend on the estimated covariance parameters of the linear mixed-effects model, and on the number of observations registered under the respective patient. It is generally closer to the overall mean than the naïve mean. This fact can be observed for

all the patients represented on Figure 9.2 without exception. It is important to mention that the estimation of the shrunk individual treatment effects labelled “shrunk mean” in Figure 9.2 involves all the data in the sample; while the naïve individual treatment effects labelled “naïve mean” on the same Figure 9.2 are estimated from the data of the respective patient only. Since shrunk individual treatment effects are estimated from more information than naïve individual treatment effects, the former are expected to be more efficient than the latter. Due to this fact, shrunk estimation of individual treatment effects through appropriate linear mixed-effects models shall be preferred overall more naïve estimation methods.

So far, the patients have been identified by numbers. However, it might be preferable to refer to the patients by their actual real names. Particularly during the later stages of the clinical trial when blinding is no longer considered a threat to biasedness. Referring to the patients by their real names might be useful when the purpose of the series of n-of-1 clinical trials is to individualize treatments to patients. In that case, names can be assigned to the patient numbers and any “trellis” object can later be updated with the desired patient names. To proceed at first a vector of names is saved for posterior use.

```
> # define patient names
> patient_names <- c(
+   "James", "Mary", "John",
+   "Patricia", "Robert", "Linda",
+   "Michael", "Barbara", "William",
+   "Elizabeth", "David", "Jennifer"
+ );
```

The twelve patient names presented here are fictitious names based on very common English language first names, and are meant to serve as an example. After saving the patient name vector, the “trellis” object used to create the plot of Figure 9.2 can be updated with the patient names as in the following lines of code.

```
> # update trellis object with patient names and plot
> trellis_02 <- update(
+   trellis_01,
+   strip=strip.custom(factor.levels=patient_names)
+ );
> print(trellis_02);
```

Note how the “update” function is used to update the “trellis_01” object with the “patient_names” vector holding the patient names. Here the “update” call returns an updated “trellis” object, which is saved under the name “trellis_02”. The print of the updated object leads to the graph presented on Figure 9.3 below.

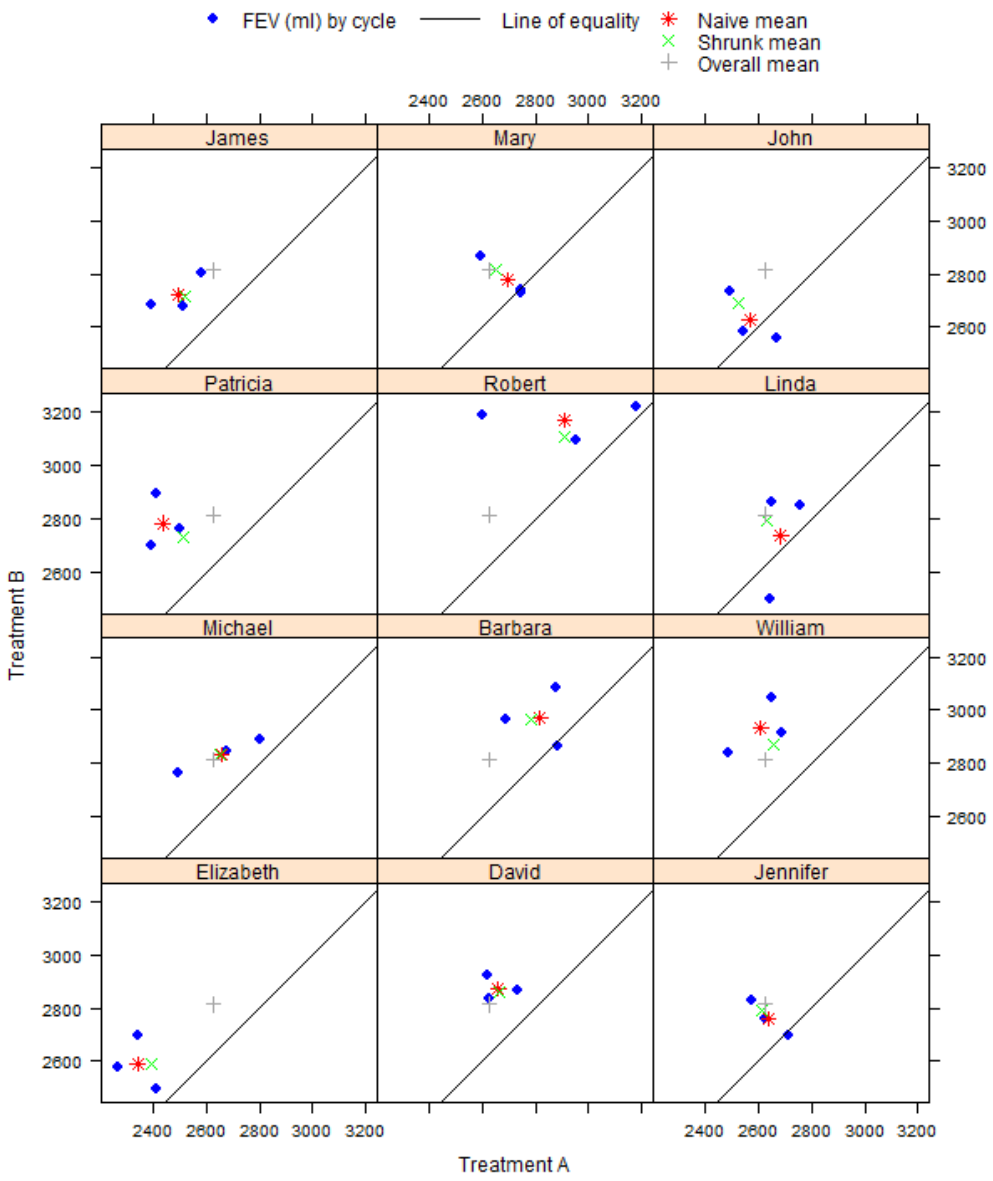


Figure 9.3: Outcome under treatment "B" versus outcome under treatment "A" for each patient.

The plot of Figure 9.3 is essentially the same as the one presented on Figure 9.2, with the exception that the trellis panels are identified by patient names instead of patient numbers.

An alternative and perhaps more convenient way of working with patient names instead of patient numbers is to rename the "Patient" factor levels within the data.frame as in the following example.

```
> # rename factor levels
> named_ddata <- ddata;
> levels(named_ddata$Patient) <- patient_names;
```

On this example, the "ddata" dataset is copied over to a new "named_data" dataset. Then the levels of the "Patient" factor within this "named_data" dataset are set to the "patient_names" vector, which

holds the actual patient names. To check that the factor levels of this new dataset are correctly set, in the next example the first and last observations are printed on screen.

```
> # print first 6 rows of data.frame
> head(named_ddata, n=6);
  Patient Cycle   YA   YB  dY
1   James     1 2394 2686 292
2   James     2 2515 2675 160
3   James     3 2583 2802 219
4    Mary     1 2746 2726 -20
5    Mary     2 2592 2867 275
6    Mary     3 2743 2742  -1
> # print last 6 rows of data.frame
> tail(named_ddata, n=6);
    Patient Cycle   YA   YB  dY
31    David     1 2617 2923 306
32    David     2 2629 2832 203
33    David     3 2732 2866 134
34 Jennifer     1 2627 2759 132
35 Jennifer     2 2712 2698 -14
36 Jennifer     3 2572 2826 254
```

Observe that the values of the “Patient” factor in the new data.frame are as desired. The same process can be applied to the full dataset “ndata”.

```
> # rename factor levels
> named_ndata <- ndata;
> levels(named_ndata$Patient) <- patient_names;
```

If these datasets are supplied to any of the plot functions accessible through the R software, the patient names instead of their respective numbers are displayed on the plots if applicable.

A trellis scatterplot of the outcome variable versus the cycle for each patient is presented in the example that follows. Before calling “xyplot” to proceed with the creation of the trellis scatterplot a legend must be defined. Since data points of the outcome for each cycle are to be plotted, there is the need to identify the treatment under which each of the data points have been registered. Having this in mind the labels in the legend must be set to the treatment labels.

```
> # define legend
> mykey_03 <- list(
+   space="top", # put legend at the top of the plot area
+   points=list(cex=1, pch=1, col="blue"),
+   text=list(
+     labels=paste0("Treatment ", levels(named_ndata$Treatment)[1]),
+     cex=1
+   ),
+   points=list(cex=1, pch=3, col="red"),
+   text=list(
+     labels=paste0("Treatment ", levels(named_ndata$Treatment)[2]),
+     cex=1
+   )
+ );
```

The legend is to be placed at the top of the plot area as specified in the “space” parameter. In this legend, the elements are to be laid out in a table of four columns and one row. The elements are a point, followed by text, again a point, and finally a text. The first point is a “blue” character of type “1”. The text that identifies this point is the value of the reference treatment on the “named_ndata” dataset. The other point is a “red” character of type “3”. Moreover, this last point’s label is the value of the test treatment on the same “named_ndata” dataset.

The “xyplot” call is similar to the ones presented in the previous examples with some minor modifications.

```
> # save and print trellis data
> trellis_03 <- xyplot(
+   Y~Cycle|Patient, # plot 'Y' vs 'Cycle' for each level of 'Patient'
+   data=named_ndata, # dataset
+   groups=Treatment, # identify 'Treatment' observations
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list( c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3) ),
+   pch=c(1, 3), # character type vector
+   col=c("blue", "red"), # color vector
+   cex=1, # character size
+   xlab="Cycle", # x axis label
+   ylab="FEV (ml)", # y axis label
+   key=mykey_03
+ );
> print(trellis_03);
```

Now the full dataset with named patients is supplied to the “data” parameter. As usual, the “Patient” factor is used as conditioning variable. The response variable is now “Y” and the explanatory variable is now “Cycle” within the “named_ndata” dataset. By specifying the “Treatment” factor in the “groups” argument a distinction between the points is requested. In this case, “xyplot” assigns different colours to the points depending on their respective treatment. By specifying col=c(“blue”,

"red"), points under the first treatment are plotted in "blue" and points under the second treatment are drawn in "red". It is also important to mention that the "pch" argument used here specifies different character types to points falling under one or the other treatment. To identify the points, the "col" and "pch" parameters in the legend list are the same as the same parameters in the "xyplot" call. Since a dataset with patient names instead of patient numbers was used in the call, the panels are expected to be labelled accordingly. The print of the "trellis_03" object returned by the "xyplot" call creates the plot presented on Figure 9.4.

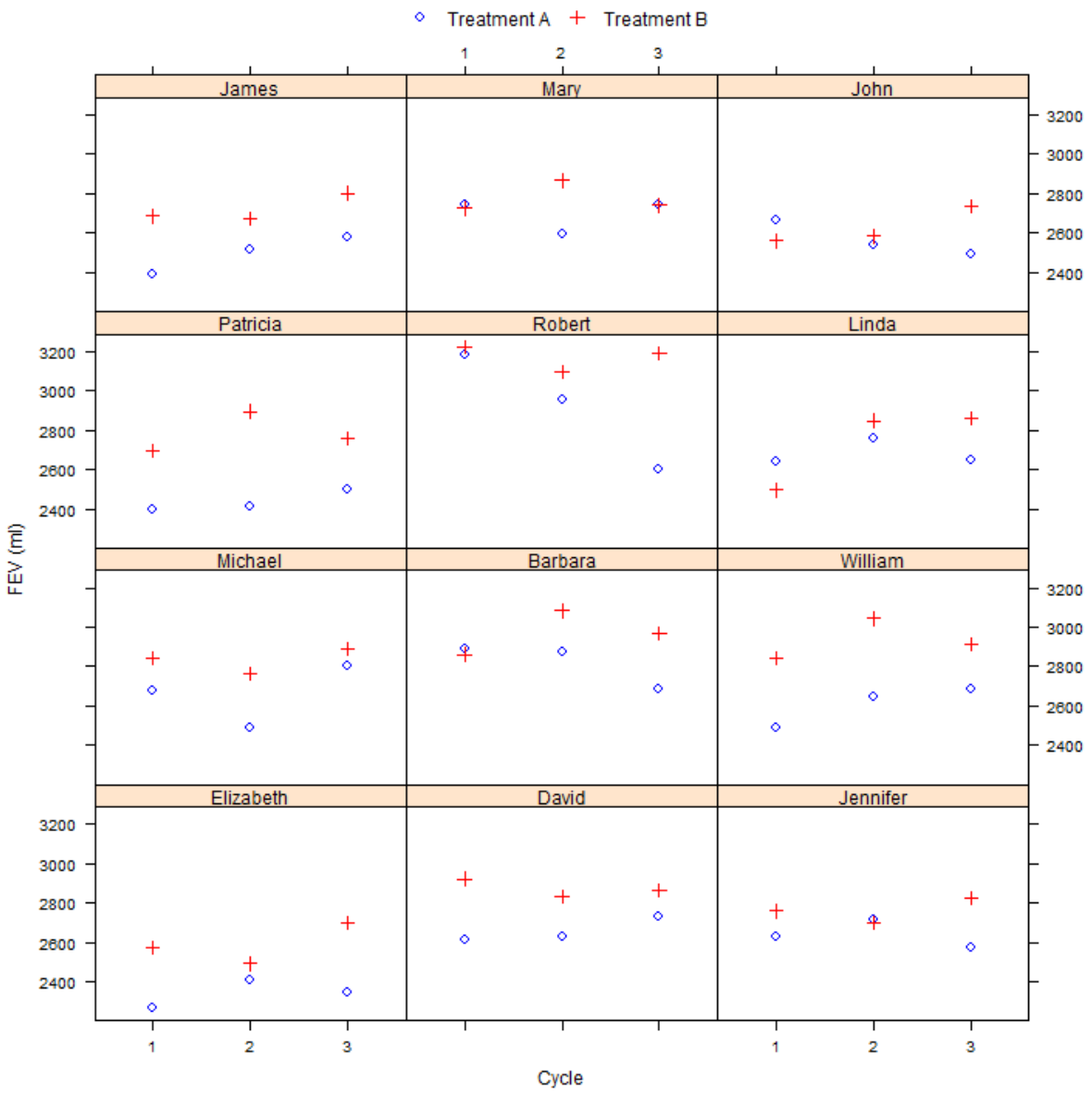


Figure 9.4: Outcome versus Cycle for each patient.

The plot presented in Figure 9.4 suggests that the outcome variable increases when the treatment is switched from "A" to "B" for all the twelve patients. The same phenomenon is observed on previous plots. For some patients there is an evident increase or decrease of the outcome variable as the trial

progresses with additional cycles. In addition, the variation of outcome variable with cycle differs from patient to patient, suggesting a cycle by patient interaction.

A scatterplot of the outcome variable difference under the two treatments versus the cycle can be obtained with a single “xyplot” call.

```
> trellis_04 <- xyplot(
+   dY~Cycle|Patient,
+   data=named_ddata,
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list( c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3) ),
+   type="p", # type of plot 'p' for points
+   cex=1, # character size
+   pch=16, # character 16
+   col="black", # color 'black'
+   xlab="Cycle",
+   ylab="Difference FEV(ml)"
+ );
> print(trellis_04);
```

This “xyplot” call is simpler than the above calls. There is only one type of point being plotted, so the definition of a legend is not required. The continuous variable “dY” is expressed in function of the categorical variable “Cycle” conditioned on the “Patient” categorical variable. The variables are looked for in the “named_ddata” dataset. A “trellis” object is returned and saved under the name “trellis_04”. Figure 9.5 is created from the print of this “trellis_04” object.

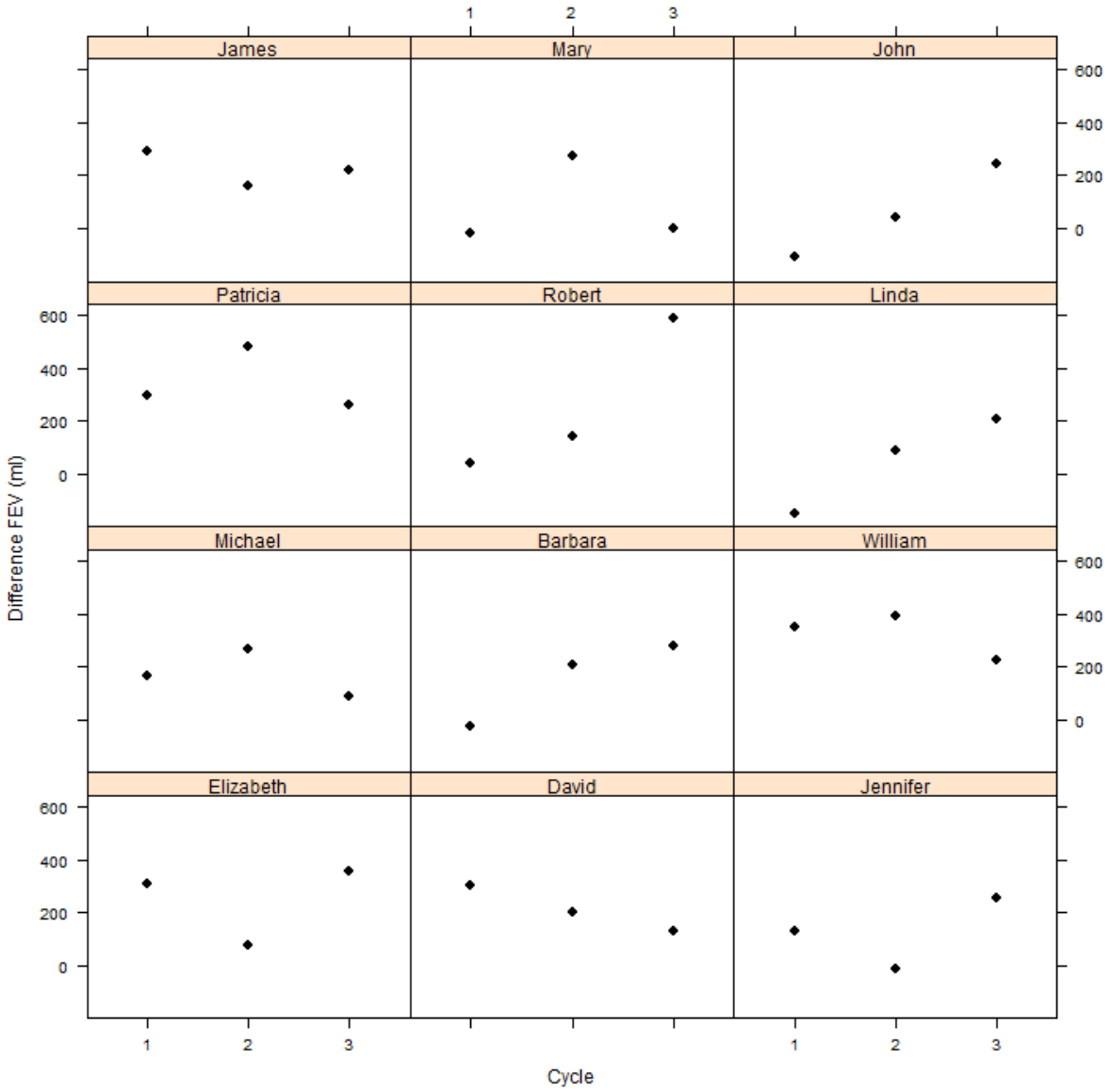


Figure 9.5: Difference outcome versus Cycle for each patient.

When interpreting the plot of Figure 9.5 it is important to remember that the outcome variable under treatment "B" minus the outcome variable under treatment "A" is represented in the ordinates axis. Thus when the points fall above the imaginary horizontal line that passes through the origin, higher values of outcome were registered under treatment "B". Therefore, the plot suggests that on average higher values of outcome are observed under treatment "B" for all the patients without exception. There is nothing new up to this point. The difference outcome varies with cycle for every patient studied. Moreover, the cycle trend differs between the patients recruited into the trial. For some patients there is an increase of difference outcome with cycle, while for others there is a decrease.

9.2 Boxplots

A boxplot is a compact and non-parametric graphical representation of the data [16]. Within this graphical representation five quantiles of the data are represented; the minimum, first quartile, median, third quartile, and the maximum. The first quartile and the third quartiles form the lower and upper sides of a rectangle or box. The median lies between the first and third quartiles of the data, so it is represented inside the box through either a point or a line extending from one side of the box to the other. To indicate variability outside the upper and lower quartiles, lines extending from the box to the minimum and maximum are drawn. Outliers may be plotted as individual points. Boxplots allow for an observation of the dispersion and skewness of the data. Longer rectangles indicate a higher interquartile range and consequently higher dispersion of the data. If the median line or point is centrally located inside the box and the lines extending from the box to the maximum and minimum have approximately equal lengths, then a symmetric distribution of the data is suggested. If it is the case that the median line or point is closer to one of the sides of the box than the distribution of the data is skewed. Boxplots are a convenient way of representing the distribution of the data. Because of their compactness, they are sometimes preferred over density plots. Boxplots of different subgroups of the data can be conveniently placed side by side allowing for a comparison between the subgroups.

In series of n-of-1 trials, the main objective is to compare treatments within and between individuals. Therefore, in this case it is more interesting to draw boxplots of the outcome variable per treatment and patient. Trellis boxplots can be obtained within the R software through the “bwplot” function provided by the “lattice” extension.

```
> # save and print trellis data
> trellis_05 <- bwplot(
+   Y~Treatment|Patient, # formula
+   data=named_ndata, # dataset
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list( c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3) ),
+   ylab="FEV (ml)", # y-axis label
+   par.settings=list(
+     box.umbrella=list( col=c("blue", "red") ), # umbrella colour
+     box.dot=list( col=c("blue", "red") ), # dot colour
+     box.rectangle=list( col=c("blue", "red") ) # rectangle colour
+   )
+ );
> print(trellis_05);
```

In this example, the “Y” outcome variable is expressed in function of the “Treatment” factor. The conditioning of this relationship on the “Patient” factor variable leads to a panel per patient totalling

twelve panels. A boxplot for each of the two treatments is expected for each patient. The colours of the boxplots are distinctively defined for the two treatments. The colours of three different parts of the boxplot can be defined independently as a list of parameters supplied to the “par.settings” argument. The colours of the “rectangle”, the “dot” and the “umbrella” parts are set equally so that the whole boxplot exhibits the same colour. In this case, boxplots of data registered under treatment “A” are blue, and boxplots of data registered under treatment “B” are red. This plot does not require the definition of a legend because the subgroup distinction is already evident. When the “trellis_05” object is printed, the graph presented in Figure 9.6 is created.

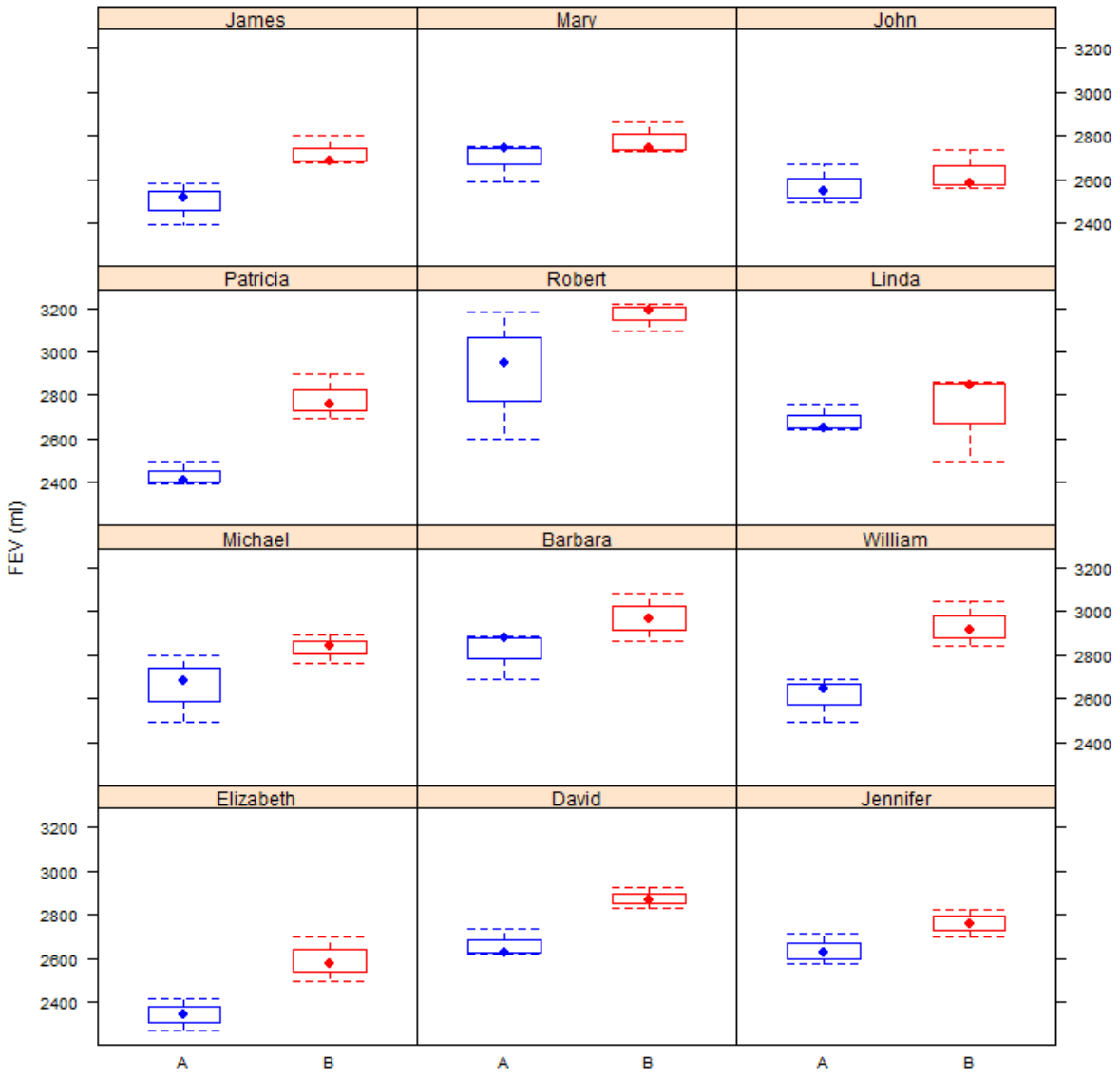


Figure 9.6: Boxplots of outcome versus treatment for each patient.

A comparison of the distribution of the data within the patients can be easily made from the observation of the plot of Figure 9.6. However, comparisons between the patients can be extremely

difficult. Therefore, it is preferable to plot all the boxplots side by side. This can be accomplished through an appropriate formula as in the following example.

```
> # save and print trellis data
> trellis_06 <- bwplot(
+   Y~Patient:Treatment, # formula
+   data=ndata, # dataset
+   xlab="Patient and Treatment", # x-axis label
+   ylab="FEV (ml)", # y-axis label
+   horizontal=FALSE,
+   par.settings=list(
+     box.umbrella=list( col=c("blue", "red") ), # umbrella colour
+     box.dot=list( col=c("blue", "red") ), # dot colour
+     box.rectangle=list( col=c("blue", "red") ) # rectangle colour
+   )
+ );
> print(trellis_06);
```

Note the difference in the formula. There is no variable conditioned on. The “Y” variable is expressed in function of the “Patient” and “Treatment” variables with a colon in between. This formula expresses the outcome variable in function of a patient by treatment interaction. The absence of the “layout” and “index.cond” in this “bwplot” call is a notorious difference in relation to the above call. There is no conditioning variable defined, which implies the absence of panels, therefore panel related parameters are not required. By setting “horizontal=FALSE” the boxplots are displayed vertically. The colours of the boxplots are kept equal to the previous “bwplot” call. After printing the “trellis_06” object, the plot of Figure 9.7 is obtained.

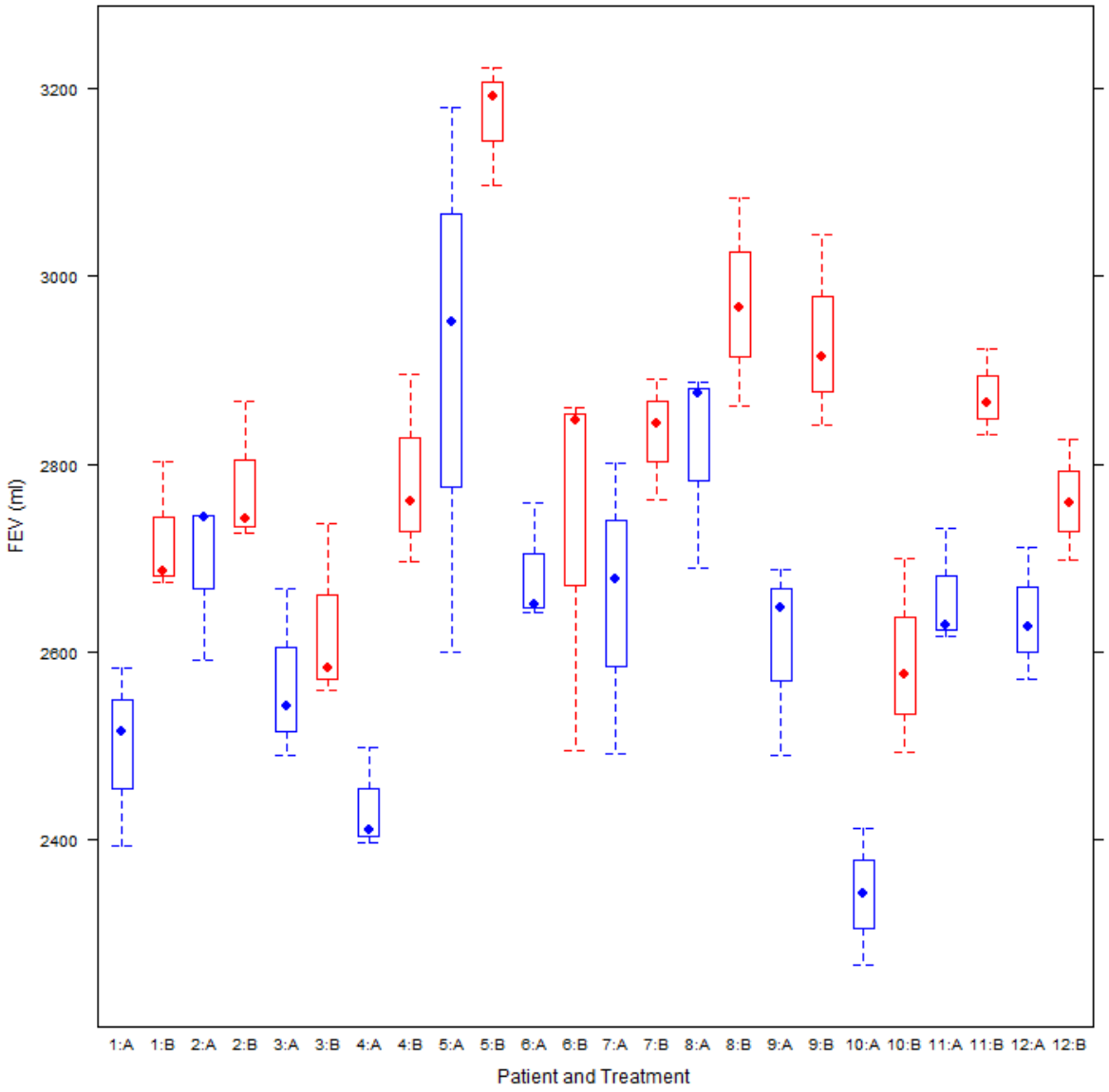


Figure 9.7: Boxplots of outcome for each patient and treatment.

Between patient comparisons of boxplots is now easier with all of them placed side by side. With the patient and treatment properly identified in the abscissa axis; and with all the boxplots related to treatment "A" in blue and all the boxplots related to treatment "B" in red; the interpretation of the data is eased. It can be observed that the outcome variable is superior under treatment "B" for every patient without exception. The plot suggests that the variation of the outcome variable is approximately equal for every patient and treatment; with notable exceptions for patient "5" and treatment "A", patient "6" and treatment "B", and patient "7" and treatment "A", where the variance of the outcome is higher than the average. Figure 9.7 also suggests that the data is skewed for a great number of patients under one treatment or the other. However, caution is advised when interpreting the plot of Figure 9.7. Note that each of the boxplots represents only three observations of the

outcome variable. With such a low number of observations, the distribution of the data shall not be considered well estimated. However, this type of plot might be very useful when applied to series of n-of-1 trials with a reasonable number of observations per patient and treatment.

If you try to use the “named_ndata” dataset instead of the “ndata” dataset to have the actual patient names identified in the abscissa axis as in the plot of Figure 9.7, you will find that the patient and treatment labels overlap to adjacent labels, making the identification of the actual boxplots extremely difficult. This problem can be overcome by plotting the outcome variable in the abscissa axis and the patient and treatment in the ordinate axis, with the boxplots drawn horizontally instead of vertically.

```
> # save and print trellis data
> trellis_07 <- bwplot(
+   Patient:Treatment~Y, # formula
+   data=named_ndata, # dataset
+   xlab="FEV (ml)", # x-axis label
+   ylab="Patient and Treatment", # y-axis label
+   horizontal=TRUE,
+   par.settings=list(
+     box.umbrella=list( col=c("blue", "red") ),
+     box.dot=list( col=c("blue", "red") ),
+     box.rectangle=list( col=c("blue", "red") )
+   )
+ );
> print(trellis_07);
```

Note that the patient by treatment interaction is expressed in function of the outcome variable in the formula. The dataset with named patients is to be used for the plot. The x-axis and y-axis labels are also defined accordingly. In this case, the “horizontal” argument is set to “TRUE” and the boxplots are drawn horizontally. The plot of Figure 9.8 is created from the code presented.

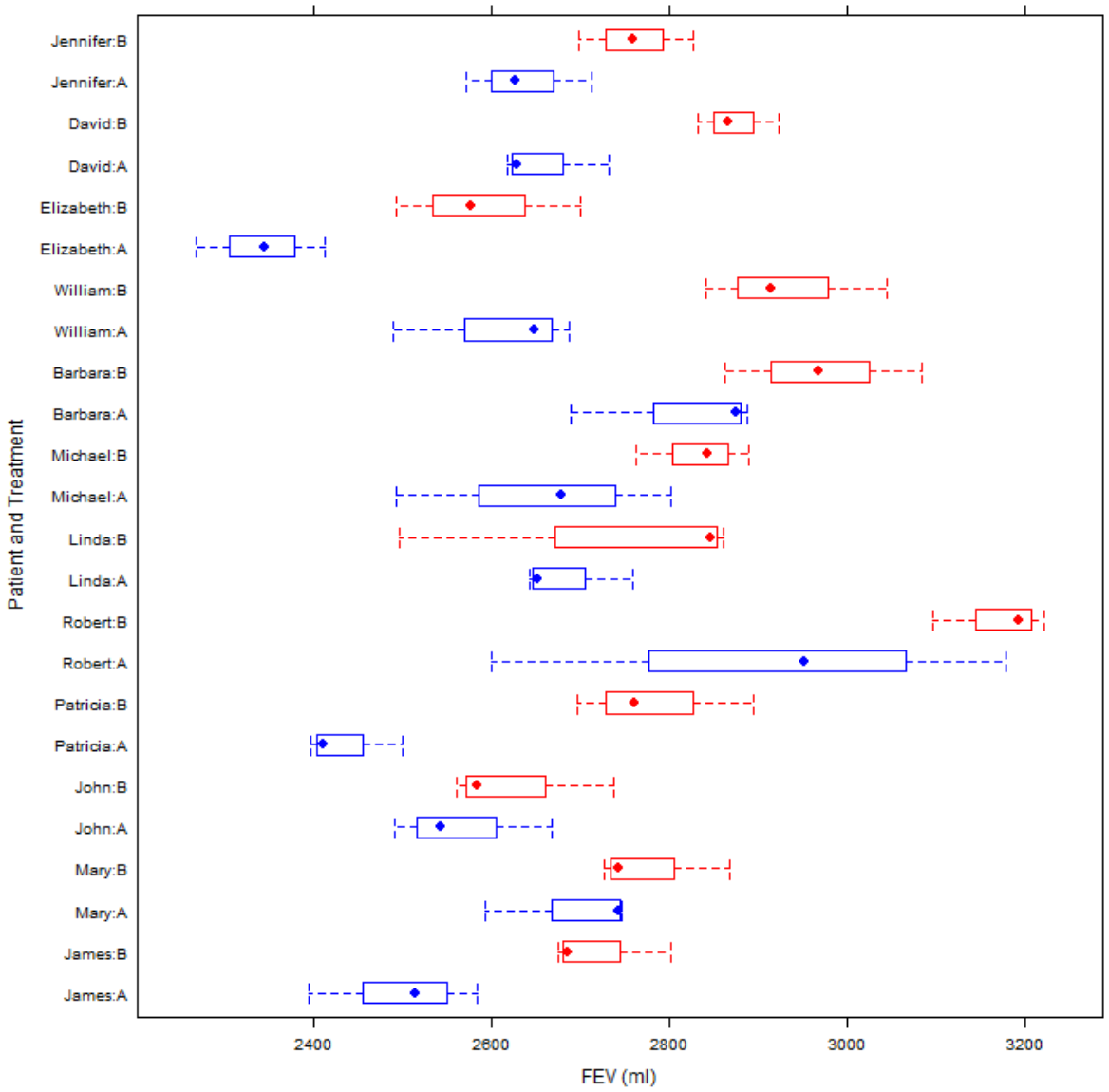


Figure 9.8: Boxplots of outcome for each patient and treatment.

Note that the patient names and treatments are well identified in the vertical axis of the plot of Figure 9.8. All the others aspects of the plot of Figure 9.8 are identical to the plot of Figure 9.7. The data is the same and the interpretations made in respect to the plot of Figure 9.7 are hence the same.

Boxplots of difference outcome for each patient can be obtained by supplying the dataset of differences to the “data” argument of “bwplot”, and defining an appropriate formula.

```
> # save and print trellis data
> trellis_08 <- bwplot(
+   Patient~dY, # formula
+   data=named_ddata, # dataset
+   xlab="Difference FEV (ml)", # x-axis label
+   ylab="Patient", # y-axis label
+   horizontal=TRUE
+ );
> print(trellis_08);
```

The “Patient” factor is defined as a function of the “dY” outcome variable, and the boxplots are plotted horizontally. Figure 9.9 below is produced after running this code.

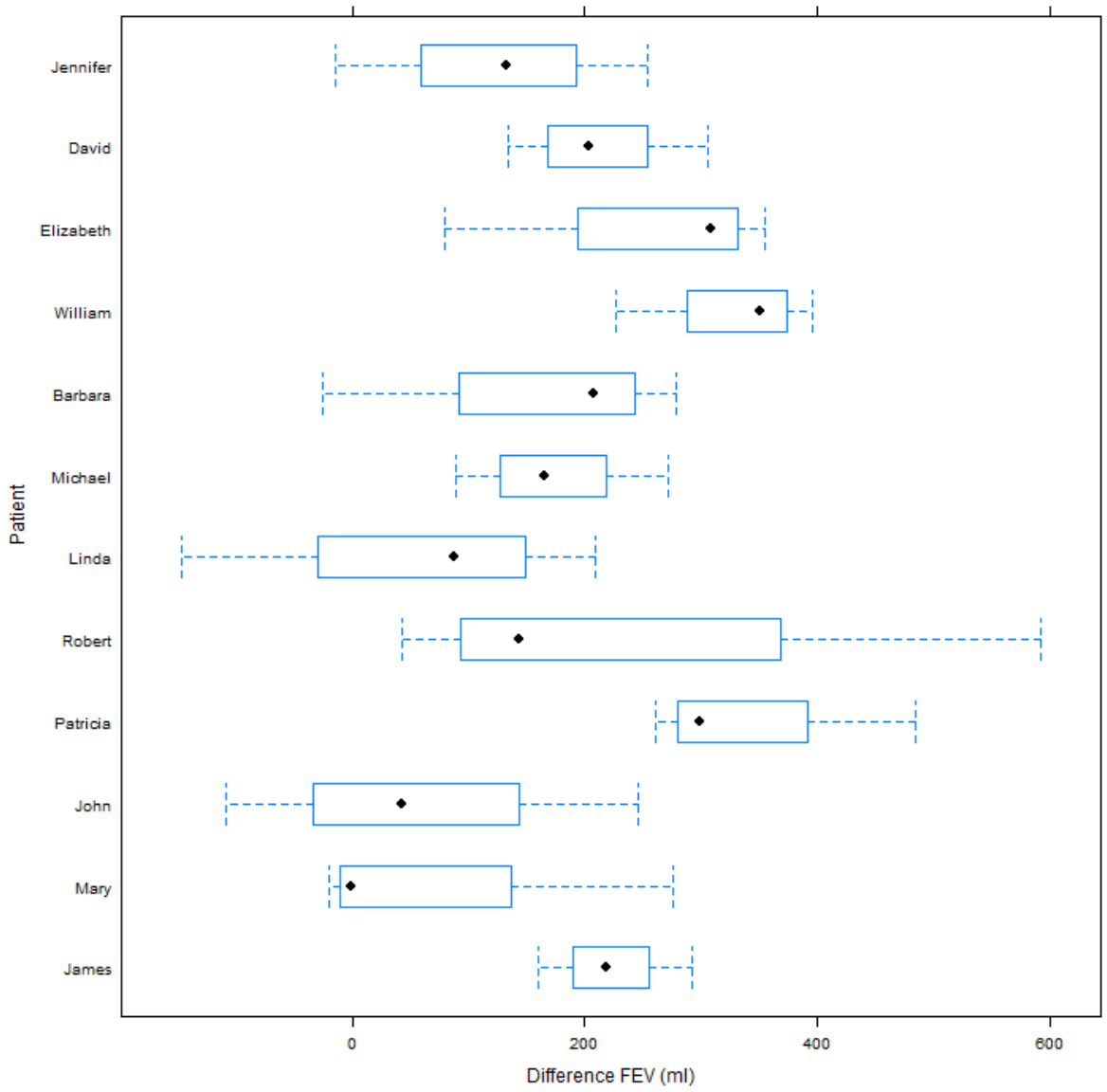


Figure 9.9: Boxplots of outcome difference for each patient.

Interpretations are identical to the ones made from Figure 9.7 and Figure 9.8. However here there are no boxplots of two treatments to compare. Instead, a vertical imaginary line passing through zero must be considered as a reference. Remember that the difference outcome variable is computed by subtracting the outcome variable under treatment “A” from the outcome variable under treatment “B”, both measured under the same cycle. The majority of the boxes as well as the medians lie above the imaginary vertical line passing through zero suggesting that higher values of outcome variable are observed under treatment “B” rather than treatment “A”. As an exception, a median very close to zero is observed for the patient “Mary”.

When plotting boxplots the median is drawn by default. To plot the mean, the definition of a panel function is required. In the following example, a panel function is defined first. This panel function draws the boxplots and plots the mean points for each patient.

```
> # define panel function
> mypanel_09 <- function(
+   x,
+   y,
+   ...
+ ) {
+   panel.bwplot(
+     x,
+     y,
+     pch="|",
+     ...
+   );
+   mean.values <- tapply(
+     X=x,
+     INDEX=y,
+     FUN=mean
+   );
+   panel.points(
+     x=mean.values[y],
+     y=y,
+     cex=1, # character size
+     pch=1, # character 1
+     col="black", # color 'black'
+     ...
+   );
+ } # mypanel_09
```

In the “mypanel_09” function, the default “bwplot” panel function is called. Here note the “pch” argument to “panel.bwplot”. According to this argument, the median is to be represented by a vertical line instead of a point as in the above examples. The coordinates of the outcome are passed through the argument “x”. In addition, the coordinates of the patient are passed through the “y” argument. After the call to “panel.bwplot”, the mean for each patient is computed through a call to “tapply”.

Finally, “panel.points” plots the mean points for each patient. The next step is the actual “bwplot” call that makes use of the defined panel function.

```
> # save and print trellis data
> trellis_09 <- bwplot(
+   Patient~dY, # formula
+   data=named_ddata, # dataset
+   panel=function(x, y, ...) {
+     mypanel_09(x, y, ...);
+   }, # panel function
+   xlab="Difference FEV(ml)", # x-axis label
+   ylab="Patient", # y-axis label
+   horizontal=TRUE
+ );
> print(trellis_09);
```

This “bwplot” call is similar to the one leading to the plot of Figure 9.9, the only difference being the use of the “mypanel_09” function. The plot of Figure 9.10 is obtained as a result.

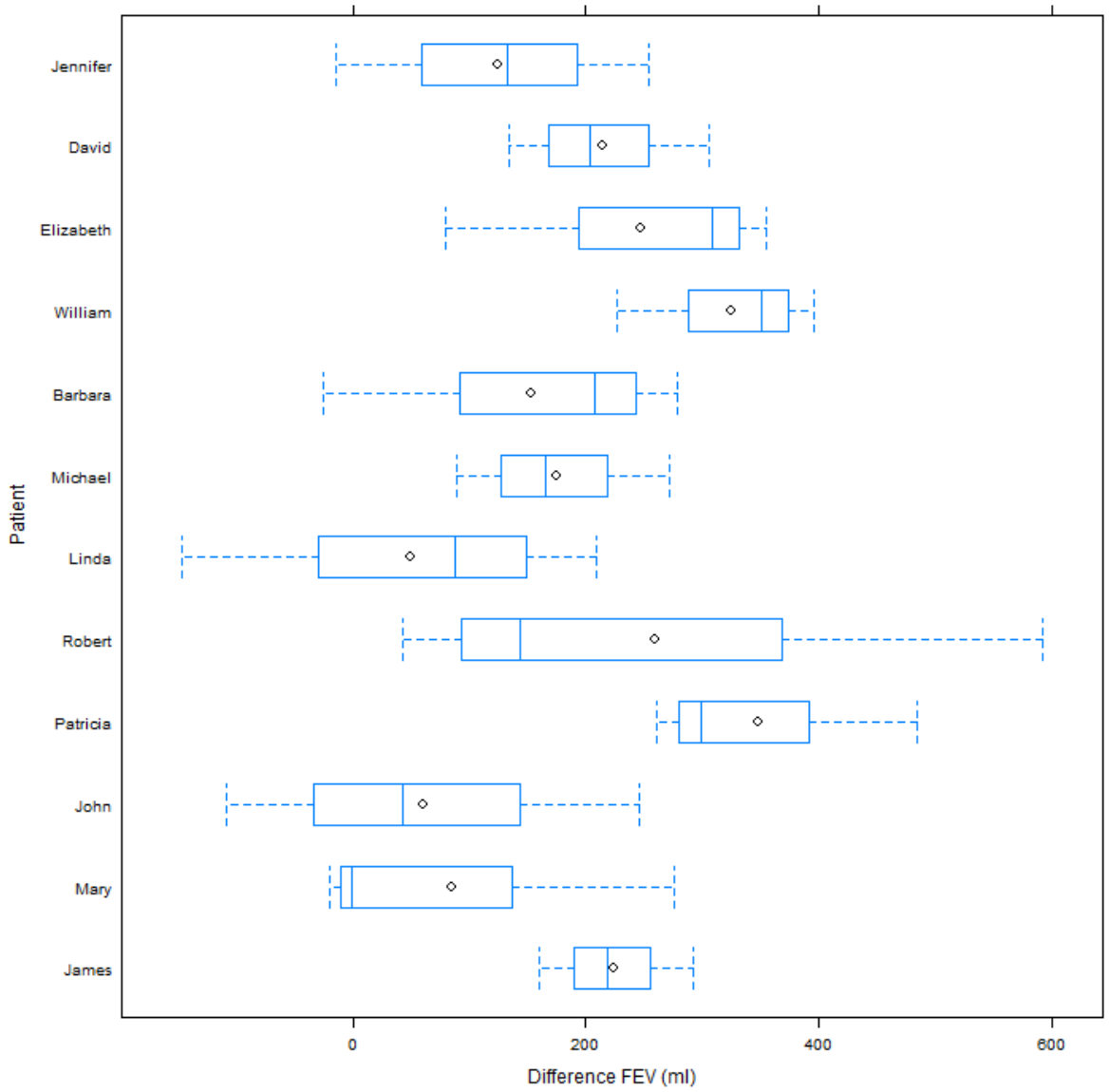


Figure 9.10: Boxplots of outcome difference for each patient.

Note that on Figure 9.10 the median is represented by a vertical line, and the mean is indicated by a small black circular area. The dataset and formula is the same as the ones used to make the plot of Figure 9.9 and for that reason, the interpretations are similar.

9.3 Density plots

The density function is a very important concept in probability theory and in Statistics in general. The density function characterizes the distribution of the data. Density plots allow the analyst to compare the distribution of the data between subgroups of the same dataset. On the other hand, the observation of density plots might suggest a very well-known distribution such as the Normal distribution. For this reason, density plots are commonly used to check model assumptions. For

example, the linear mixed-effects models presented in chapters 7, and 8 assume that the errors are normally distributed within and between groups. Therefore, density plots of the errors obtained from the model provide a way of checking that assumption, given that the Normal density assumes a very well-known symmetric bell shaped curve. However, the purpose here is not to cover model assumption checking techniques. It is the purpose of this document to demonstrate some of the graphical facilities available within the R software for displaying data arising from series of n-of-1 trials. To determine the curve for plotting, kernel density estimation is used. Kernel density estimation is a non-parametric way of estimation of the probability density function of the data. Kernel density estimation is a very wide field that cannot be covered here. The texts of Silverman [17], Wand and Jones [18], Bowman and Azzalini [19] and Scott [20] are suggested to those interested to know more about the subject.

Density plots of the data for each patient and treatment are considered next. Before making the actual plot, an appropriate legend must be defined.

```
> # define legend
> mykey_10 <- list(
+   space="top", # put legend at the top of the plot area
+   lines=list(lty="solid", lwd=1, col="blue"),
+   text=list(
+     paste("Treatment ", levels(named_ndata$Treatment)[1], sep="")
+   ),
+   lines=list(lty="dashed", lwd=1, col="red"),
+   text=list(
+     paste("Treatment ", levels(named_ndata$Treatment)[2], sep="")
+   )
+ );
```

The legend is to be placed on top of the plot area. The density curve under treatment “A” is identified with a solid blue line. Whereas the density curve under treatment “B”, is to be identified by a dashed red line. Trellis density plots can be obtained through the “densityplot” function provided by the “lattice” package. The following lines of code make use of it.

```
> # save and print trellis data
> trellis_10 <- densityplot(
+   ~Y|Patient, # formula
+   data=named_ndata, # dataset
+   groups=Treatment,
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list(
+     c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3)
+   ), # define panel order
+   xlab="FEV (ml)", # x axis label
+   ylab="Density", # y axis label
+   lty=c("solid", "dashed"), # line type
+   lwd=1, # line width
+   col.line=c("blue", "red"), # line color
+   plot.points=TRUE, # plot points
+   cex=0.6, # point symbol expansion
+   pch=c(16, 1), # point symbol
+   col=c("blue", "red"), # point color
+   key=mykey_10 # list of legend parameters
+ );
> print(trellis_10);
```

The density of the outcome variable “Y” conditioned on the patient is specified in the formula. The “groups” argument splits the density curves according to the levels of the “Treatment” factor. This leads to two density curves one for each treatment under each panel defined by the “Patient” conditioning variable. Figure 9.11 is obtained after running the code.

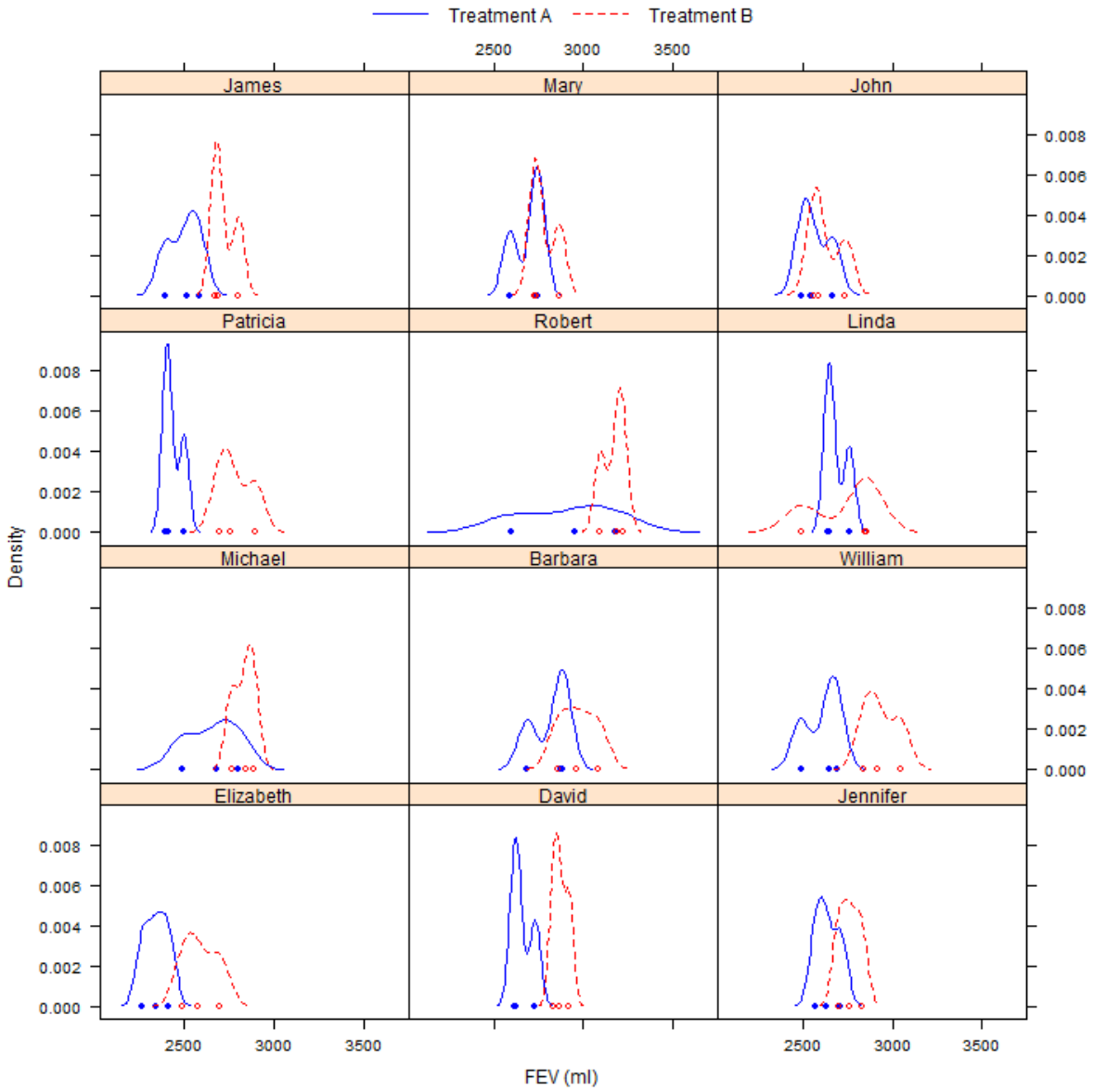


Figure 9.11: Density of outcome per treatment and patient.

The plot of Figure 9.11 suggests that higher values of outcome are observed under treatment "B" when compared to treatment "A". Given that, the density curves relative to treatment "B" lie on the right side of the density curves for treatment "A", regardless of the patient. The density curve under treatment "A" for "Robert" is wider than the other curves, suggesting that the variance is higher for this treatment and patient. All the curves exhibit a bell like shape characteristic of the normal density. Some curves suggest a mixed normal density with two modes. It is important to note however that there are only three observations available for estimation of each curve. With such a low number of points, kernel density estimation is not expected to behave well and there is a high level of uncertainty associated with it. However, the plots exemplified might provide useful insight into n-of-1 trials designed with more periods or cycles.

It is possible to plot all the density curves side by side without resorting to conditioning. However due to the large number of curves occupying the same area there is a lot of confusion. In this case, it does not facilitate interpretation. Potting the density curves on the same plot area might be useful in other situations. For this reason, an example is presented. In this example, the dataset of outcome differences is used.

```
> trellis_11 <- densityplot(
+   ~dY,
+   data=named_ddata,
+   groups=Patient,
+   xlab="Difference FEV(ml)", # x axis label
+   ylab="Density", # y axis label
+   plot.points=FALSE, # do not plot points
+   ref=TRUE, # add reference line at zero
+   auto.key=list(space="top", columns=4)
+ );
> print(trellis_11);
```

The formula is fairly simple in this case. Only the outcome variable is specified in the formula. The “Patients” variable in the “groups” argument, leads to one density curve for each patient. If this argument is not specified than a unique density curve pertaining to the whole dataset is drawn. Plotting the points is not very useful either. Due to the large number of points falling over the same region, a distinction between them is rather difficult. Specifying “plot.points=FALSE” prevents the points from being drawn. The “ref=TRUE” argument requests a horizontal line at zero. Unlike in previous examples, an extensive list of legend parameters is not supplied to the function call. Instead, a list with a minimal set of parameters is supplied to the “auto.key” parameter. In this case, the legend is to be placed in the top of the plot area. The labels are to be laid out in four columns. The legend is automatically defined from the levels of the “Patient” factor defined in the “groups” parameter. The graph is present on Figure 9.12 below.

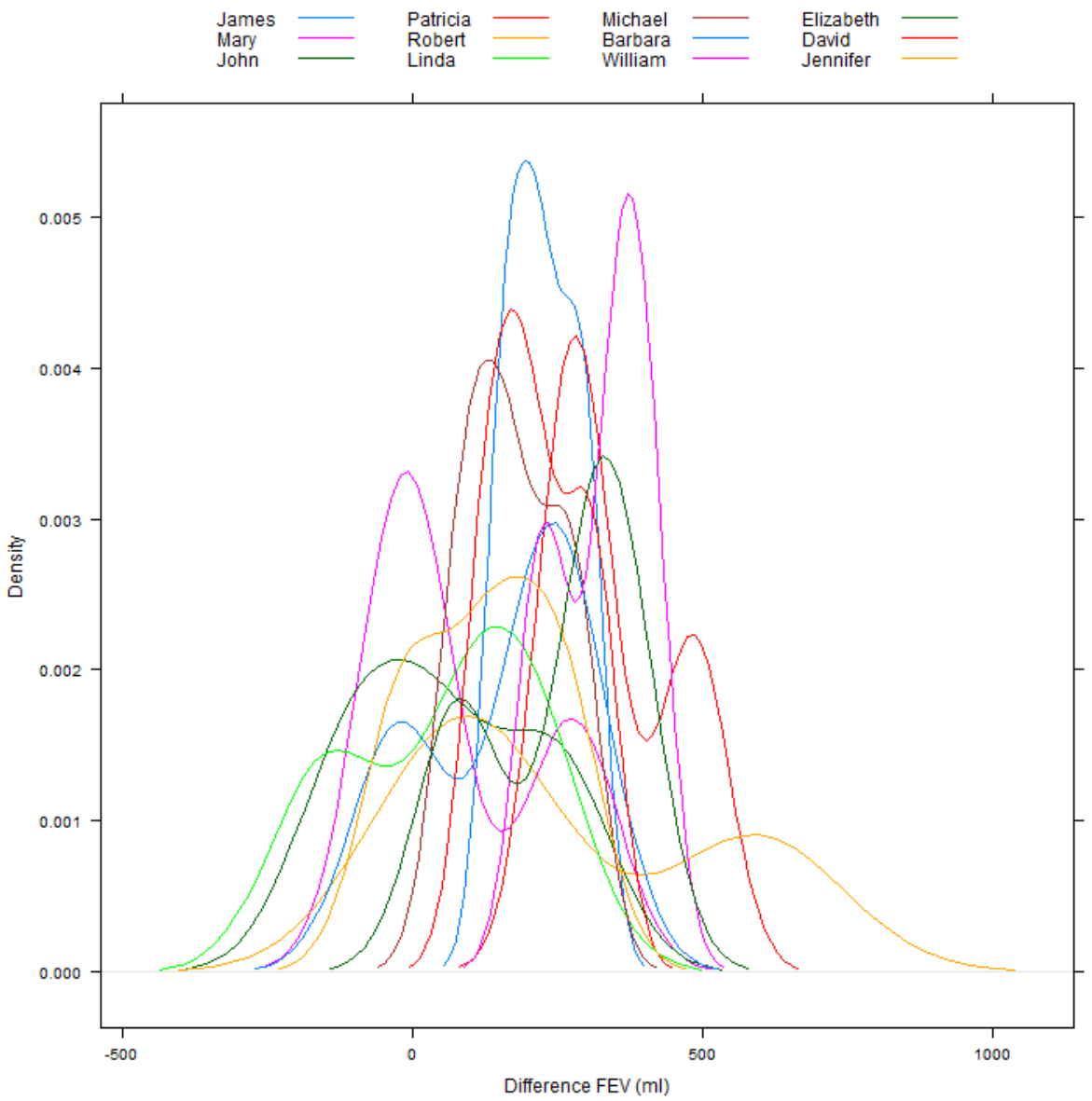


Figure 9.12: Density of difference outcome for each patient.

A plot like the one presented in Figure 9.12 can be used to perform a between patient comparison of the density curves of the outcome difference under two treatments. As mentioned before the number of curves occupying the same region of the graph makes interpretations difficult. Despite this difficulty, it can be observed that the distribution of the outcome difference shows a remarkably distinct mean and variance between the patients.

Density estimation for each treatment involves more observations than estimation for each patient. Therefore, a density plot of the outcome for each treatment is more useful. The next example shows how this can be done with the R software.

```
> # define legend
> mykey_12 <- list(
+   space="top", # put legend at the top of the plot area
+   lines=list(lty="solid", lwd=1, col="blue"),
+   text=list(
+     paste("Treatment ", levels(ndata$Treatment)[1], sep="")
+   ),
+   lines=list(lty="dashed", lwd=1, col="red"),
+   text=list(
+     paste("Treatment ", levels(ndata$Treatment)[2], sep="")
+   )
+ );
>
> # save and print trellis data
> trellis_12 <- densityplot(
+   ~Y,
+   data=ndata,
+   groups=Treatment,
+   xlab="FEV (ml)", # x axis label
+   ylab="Density", # y axis label
+   lty=c("solid", "dashed"), # line type
+   lwd=1, # line width
+   col.line=c("blue", "red"), # line color
+   plot.points=TRUE, # plot points
+   cex=0.6, # point symbol expansion
+   pch=c(16, 1), # point symbol
+   col=c("blue", "red"), # point color
+   ref=TRUE, # add reference line at zero
+   key=mykey_12
+ );
> print(trellis_12);
```

The complete dataset must be used rather than the dataset of outcome differences. The formula contains only the outcome variable “Y”. In this case, the “Treatment” variable is specified in the “groups” parameter. The legend labels are determined from the levels of the “Treatment” variable. Moreover, the legend is placed on the top of the plot area. The plot is presented on Figure 9.13 below.

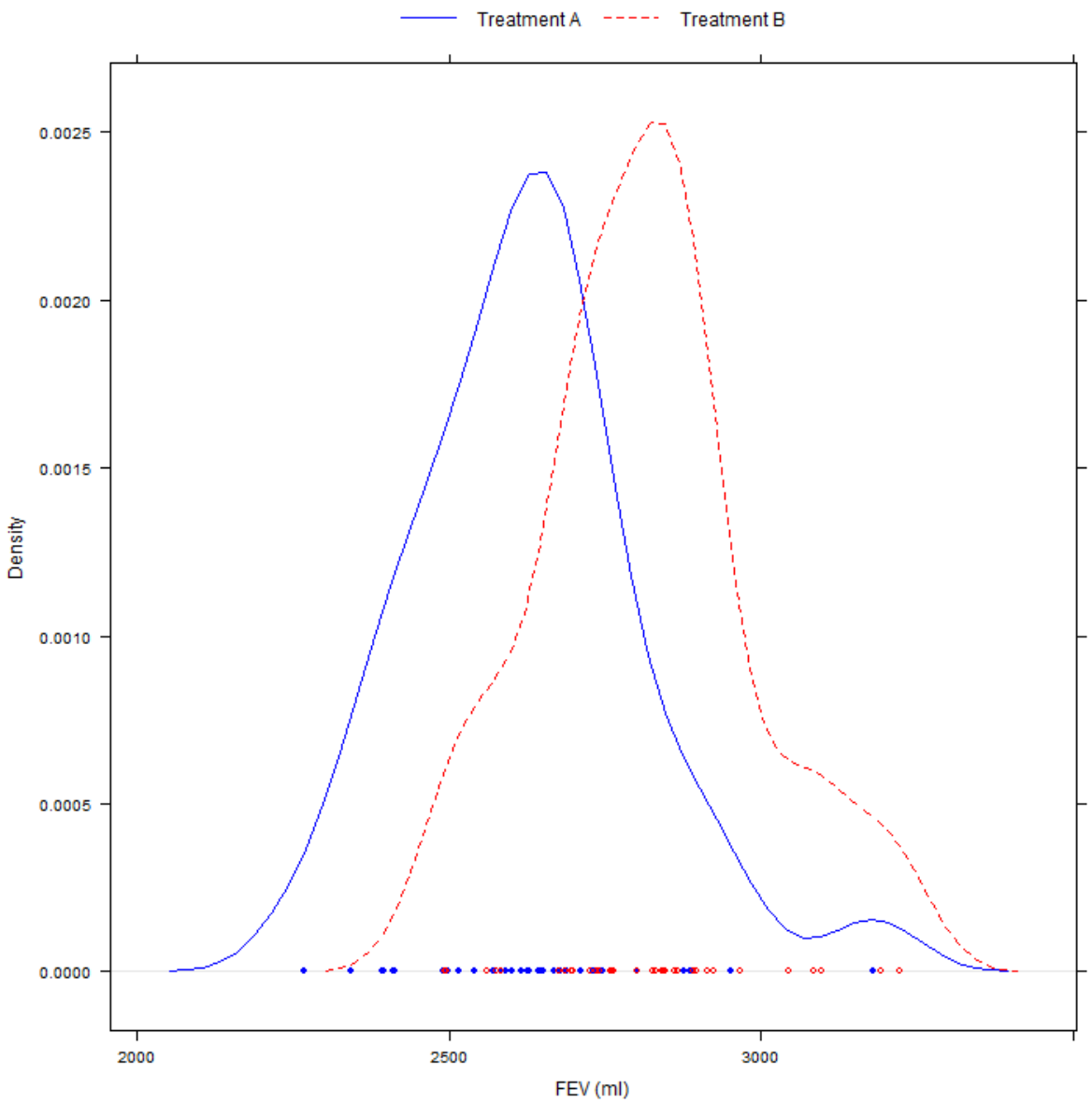


Figure 9.13: Density of outcome for each treatment.

The density curve of the data registered under treatment “B” lies on the right of the one registered under treatment “A”. This suggests that the outcome is higher on average under treatment “B”. The density curve relative to treatment “A” is wider than the curve relative to treatment “B”, suggesting that the variance is higher under treatment “A” than under treatment “B”. Both curves resemble the normal density curve, suggesting the data is approximately normal. The plot of Figure 9.13 refers to the data from all the patients. Therefore, within and between patient comparisons cannot be made.

9.4 Dot plots

A dot plot is a simple statistical chart, in which dots are used to represent data points associated with categorical variables. For data arising from series of n-of-1 trials, it makes sense to

plot the points associated with the measurements recorded on each patient. This permits within and between patient comparisons to be made. Since the main objective is to assess the difference between treatments, the points must be properly identified. Therefore, in the next examples the point character and colour are defined according to the respective treatment. One starts by defining a list of parameters for the legend.

```
> mykey_13 <- list(
+   space="top", # put legend at the top of the plot area
+   points=list(cex=1, pch=16, col="blue"),
+   text=list(
+     paste("Treatment ", levels(named_ndata$Treatment)[1], sep="")
+   ),
+   points=list(cex=1, pch=1, col="red"),
+   text=list(
+     paste("Treatment ", levels(named_ndata$Treatment)[2], sep="")
+   )
+ );
```

The legend is placed on the top of the plot area. Treatment “A” is identified by a small blue circle. While treatment “B” is identified by a small red circumference. Dot plots can be obtained from either “dotplot” or “stripplot” functions with similar results. The difference is that “dotplot” by default draws a line passing through the points while “stripplot” does not. In the following code, “dotplot” is used.

```
> # save and print trellis data
> trellis_13 <- dotplot(
+   ~Y|Patient, # formula
+   data=named_ndata, # dataset
+   groups=Treatment,
+   layout=c(3, 4), # 3 columns 4 rows
+   index.cond=list(
+     c(10, 11, 12, 7, 8, 9, 4, 5, 6, 1, 2, 3)
+   ), # define panel order
+   xlab="FEV (ml)", # x axis label
+   cex=1, # point symbol expansion
+   pch=c(16, 1), # point symbol
+   col=c("blue", "red"), # point color
+   key=mykey_13
+ );
> print(trellis_13);
```

The outcome variable “Y” is conditioned on the “Patient” variable leading to a dot plot for each patient. By supplying the “Treatment” to the “groups” argument, an identification of the points according to the treatments is requested. The legend is defined according to the “mykey_13” list. The plot is presented on Figure 9.14 below.

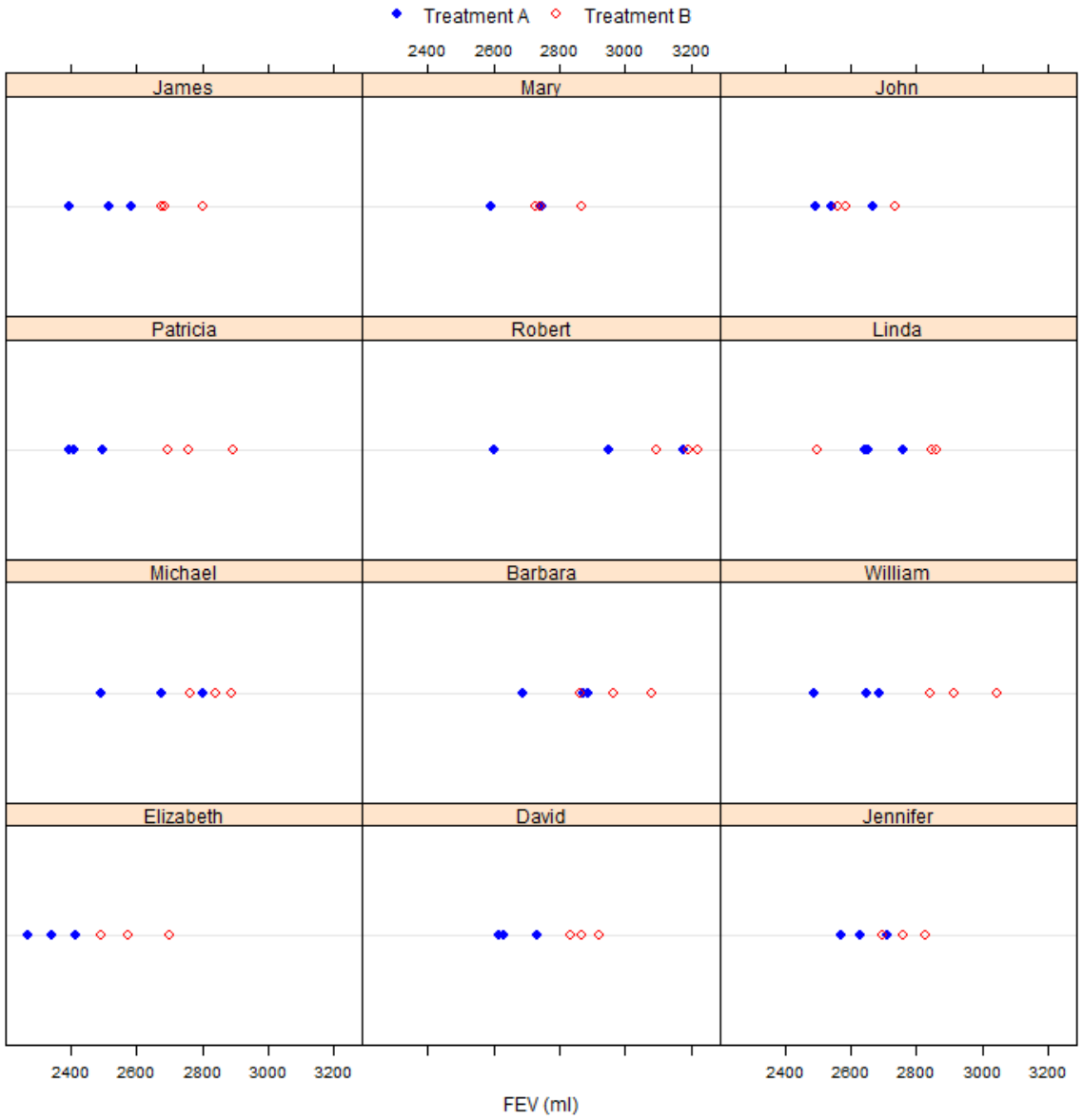


Figure 9.14: Dot plot of outcome per patient.

For all the patients without exception the majority of the points relative to treatment "B" lie on the right of the points recorded under treatment "A", suggesting that the outcome is higher under treatment "B" most of the time.

It is very difficult to perform between patient comparisons with the plot of Figure 9.14. The next example considers a dot plot where between and within patient comparisons can be made very easily. The example makes use of "stripplot". First, a list of parameters for the legend is defined.

```
> # define legend
> mykey_14 <- list(
+   space="top", # put legend at the top of the plot area
+   points=list(cex=1, pch=8, col="red"),
+   text=list(labels="Naive treatment effect", cex=1),
+   points=list(cex=1, pch=4, col="green"),
+   text=list(labels="Shrunk treatment effect", cex=1),
+   lines=list(lty="solid", lwd=1, col="gray"),
+   text=list(labels="Overall treatment effect", cex=1)
+ );
```

The legend is placed on the top of the plot area. It identifies three labels. The “Naïve treatment effect” is identified by a red star point. The “Shrunk treatment effect” is identified by a green cross point. In addition, the “Overall treatment effect” is to be identified by a grey solid line. Afterwards a panel function is defined.

```
> # define panel function
> mypanel_14 <- function(
+   x,
+   y,
+   ...
+ ) {
+   fit <- lmer(
+     formula=Outcome~1+(1|Patient),
+     data=data.frame(
+       "Patient"=y,
+       "Outcome"=x
+     ),
+     REML=TRUE
+   );
+   shrunk.values <- predict(
+     object=fit,
+     re.form=~(1|Patient)
+   );
+   mean.values <- tapply(
+     X=x,
+     INDEX=y,
+     FUN=mean
+   );
+   panel.stripplot(
+     x,
+     y,
+     type="p", # type of plot 'p' for points
+     cex=1, # character size
+     pch=1, # character 1
+     col="black", # color 'black'
+     ...
+   ); # plot data points
+   panel.points(
+     x=mean.values[y],
```

```
+     y=y,
+     cex=1, # character size
+     pch=8, # character 8
+     col="red", # color 'red'
+     ...
+   ); # plot naive mean
+   panel.points(
+     x=shrunk.values,
+     y=y,
+     cex=1, # character size
+     pch=4, # character 4
+     col="green", # color 'green'
+     ...
+   ); # plot shrunk mean
+   panel.abline(
+     v=fixef(fit)[1], # vertical line
+     lty="solid", # line type 'solid'
+     lwd=1, # line width
+     col="gray" # color 'gray'
+   ); # plot overall mean
+   panel.abline(
+     v=0, # vertical line
+     lty="dashed", # line type 'dashed'
+     lwd=1, # line width
+     col="gray" # color 'gray'
+   ); # plot reference line
+ } # mypanel_14
```

The panel function performs several tasks. It starts by fitting a linear mixed-effects model of difference to the data. Fitting of the model in question within R is referred in chapter point 8 above. It then computes the patient conditioned predicted values from the fitted model. These predicted values are the shrunk treatment effects. It follows by computing the mean of the data for each patient and considering that patient's data only. These individual mean values are the naïve treatment effects. Then the function plots the actual data points for each patient, the naïve means, the shrunk means, and a vertical solid line passing through the overall mean. Finally, it draws a dashed line passing through zero. After having access to a list of legend parameters and to a panel function, a call to "stripplot" is made.

```
> # save and print trellis data
> trellis_14 <- stripplot(
+   Patient~dY, # formula
+   data=named_ddata, # dataset
+   xlab="Difference FEV(ml)", # x axis label
+   panel=function(x, y, ...) {
+     mypanel_14(x, y, ...);
+   }, # panel function
+   key=mykey_14
+ );
> print(trellis_14);
```

According to the formula argument, the “Patient” variable is plotted in the y-axis, and the outcome difference variable “dY” is plotted in the x-axis. The dataset of outcome differences with named patients is used. In addition, the x-axis label is defined accordingly. The “panel” parameter defines a function, which calls “mypanel_14” defined above. The last parameter is the “key”, which equals the “mykey_14” list of legend parameters. The plot obtained is presented on Figure 9.15 below.

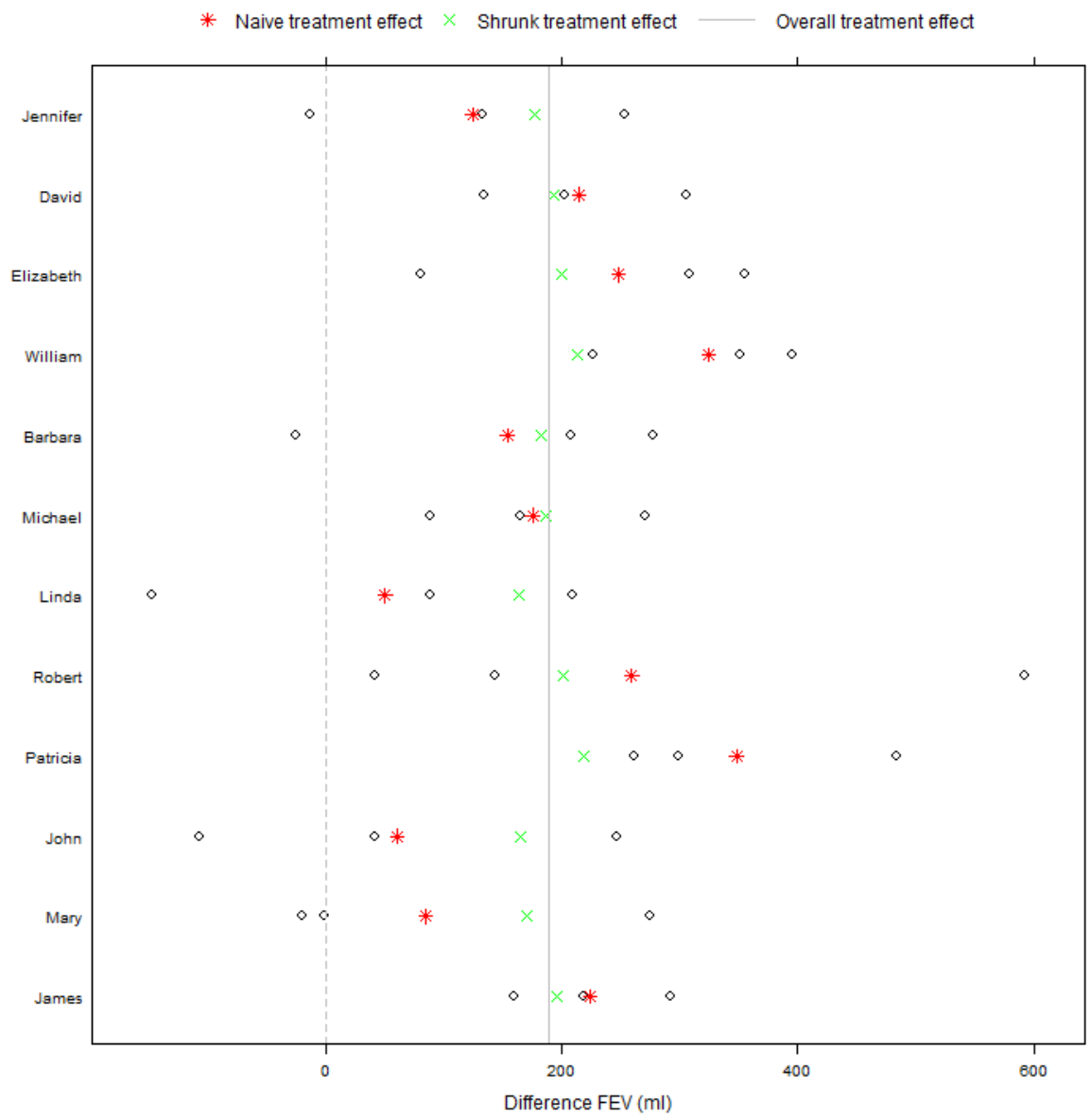


Figure 9.15: Dot plot of outcome difference per patient.

Between patient comparisons are now possible. The plot of Figure 9.15 suggests that for some patients the individual treatment effect is lower than the overall treatment effect, while there are patients for which the reverse happens. In other words, the plot of Figure 9.15 suggests the existence of a treatment by patient interaction. All the individual treatment effects, naïve and shrunk are positive, suggesting that the outcome is higher under treatment "B" than under treatment "A" on average for each patient. Therefore, if the patients benefit from higher values of outcome, then treatment "B" shall be preferred. If it is the case that lower values of outcome are preferred, then the choice shall fall over treatment "A". The shrunk treatment effects are closer to the overall mean than to the respective naïve counterparts. The shrunk treatment effects are a weighted average of the

overall mean and the naïve individual mean. The plot suggests that more weight is given to the overall mean than to the naïve individual mean in this case.

References

1. Pinheiro, J.C. and D.M. Bates, *Mixed-Effects Models in S and S-PLUS*. 2000, New York: Springer.
2. Pinheiro, J., et al., *nlme: Linear and Nonlinear Mixed Effects Models*. 2015.
3. Bates, D., et al., *Fitting Linear Mixed-Effects Models Using lme4.* 2015, 2015. **67**(1): p. 48.
4. Højsgaard, U.H.a.S., *A Kenward-Roger Approximation and Parametric Bootstrap Methods for Tests in Linear Mixed Models – The R Package pbkrtest.* Journal of Statistical Software, 2014. **59**(9).
5. Kenward, M.G. and J.H. Roger, *Small sample inference for fixed effects from restricted maximum likelihood.* Biometrics, 1997. **53**(3): p. 983-97.
6. R Core Team, *R: A Language and Environment for Statistical Computing*. 2015, R Foundation for Statistical Computing: Vienna, Austria.
7. Senn, S., *Cross-over Trials in Clinical Research*. 2nd ed. 2002, Chichester, England: John Wiley & Sons.
8. Senn, S., *Statistical Issues in Drug Development*. 2nd ed. 2007, Chichester, England: John Wiley & Sons.
9. Jones, B. and M.G. Kenward, *Design and Analysis of Cross-Over Trials*. 2nd ed. 2003, London, England: Chapman & Hall.
10. Zucker, D.R., et al., *Combining single patient (N-of-1) trials to estimate population treatment effects and to evaluate individual patient responses to treatment.* J Clin Epidemiol, 1997. **50**(4): p. 401-10.
11. Zucker, D.R., R. Ruthazer, and C.H. Schmid, *Individual (N-of-1) trials can be combined to give population comparative treatment effect estimates: methodologic considerations.* J Clin Epidemiol, 2010. **63**(12): p. 1312-23.
12. Verbeke, G. and G. Molenberghs, *Linear Mixed Models for Longitudinal Data*. 2009, New York: Springer.
13. DerSimonian, R. and N. Laird, *Meta-analysis in clinical trials.* Control Clin Trials, 1986. **7**(3): p. 177-88.
14. Viechtbauer, W., *Conducting Meta-Analyses in R with the metafor Package.* 2010, 2010. **36**(3): p. 48.
15. Sarkar, D., *Lattice: Multivariate Data Visualization with R*. 2008, New York: Springer.
16. Tukey, J.W., *Exploratory Data Analysis*. 1977: Addison-Wesley Publishing Company.
17. Silverman, B.W., *Density Estimation for Statistics and Data Analysis*. 1986: Chapman & Hall/CRC.

18. Wand, M.P. and M.C. Jones, *Kernel Smoothing*. 1994: Chapman & Hall/CRC.
19. Bowman, A.W. and A. Azzalini, *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. 1997: OUP Oxford.
20. Scott, D.W., *Multivariate Density Estimation: Theory, Practice, and Visualization*. 2015, Hoboken, New Jersey: John Wiley & Sons.